Reading and Writing Arrays across Excel and DLLs

AVt, Nov 2005

This is a short guide for reading and writing arrays either starting from VBA or from a DLL.

The examples are covered by an Excel worksheet and C source code for the DLL. For compiling a DLL with MSVC6 the option "`__stdcall`" has to be used and functions to be exported need an additional export file named *.def.

Conventions for VBA:

- all array indices start at 1,
- all arrays are of type double and are column vectors in n-space $\mathbb{R}^n$
  (i.e. array(i) = array(i,1) and all the arrays here are numerical ones)

Functions exported from the DLL have names ending with "`_DLL`" in Excel (a naming convention to make things easier to read).

All the functions are equipped with test cases, but are easily adapted for real tasks (by changing their outputs or deleting test code).

For simplicity no global VBA arrays are used. Thus all arrays are contained in functions, if it is necessary they can be copied them from there.

First we start from VBA where a function writeDLL holding data in an array arrVBA should write that array to an array arrDLL, which is locally in a DLL function receiveVBA. Conversely we consider a VBA function with a local array arrVBA, which wants to read an array arrDLL, which is located within a DLL function deliverVBA. For simplicity: VBA is a client to be served by the DLL.

The other way round will be that DLL is the client to be served by VBA (which is not quite correct, as I always want to look at the DLL through Excel) and reverse the meanings and arrows.

Hence 4 cases will be treated and schematically it looks like this:

```
              VBA                                     DLL


                          calling from VBA

   fct writeDLL with arrVBA      --->    arrDLL in fct receiveVBA
   fct readDLL  with arrVBA      <---    arrDLL in fct deliverVBA


                          calling from DLL

   fct receiveDLL with arrVBA    <---    arrDLL in fct writeVBA
   fct deliverDLL with arrVBA    --->    arrDLL in fct readVBA
```

In the latter case a trigger will be used to make things observable from Excel.

In each case the calling function provides memory which is accessed by the called function to copy data to a local array there. This is possible, since arrays are passed by reference. Within C this is just running through the memory starting at the array base. In VBA that is not directly possible, but can be done using an API call.
Within the C part we will need

```
#include <stdlib.h>
#include <memory.h>
#include <malloc.h>

#define min(a,b)    (((a) < (b)) ? (a) : (b))

typedef long (__stdcall * l_pF_al)(double*, long);
```

The last definition is needed for a call back (see below).

1. On demand of Excel write an array from VBA to C

The function wants to send its array to the DLL, where arrVBA is filled with ascending natural numbers. The receiving array sits in receiveVBA and (for testing) returns the sum of the numerical entries.

```
Function writeDLL(ByVal nData As Long)
Dim arrVBA() As Double
Dim nResult As Long
Dim i As Long

If nData < 1 Then nData = 1

' initialize array to 0.0
ReDim arrVBA(nData)

' fill array with ascending natural numbers
For i = 1 To UBound(arrVBA)
  arrVBA(i) = i
Next

' hand this array to DLL where data are written to a local array
nResult = receiveVBA_DLL(arrVBA(1), nData)

writeDLL = nResult
End Function
```

Note the calling is done by the first array element.

The receiving function within the DLL is

```
long __stdcall
receiveVBA(double *arrVBA_source, long nData)
// function holding a local array to receive data from VBA,
// arrVBA_source is the array defined within VBA
{
    double * arrDLL_destination;
    long nMax=0, nResult=0;

    arrDLL_destination = (double *)calloc(nData,sizeof(double));
    nMax = (long)( _msize(arrDLL_destination)/(double)(sizeof(double)) );

    memcpy(arrDLL_destination, arrVBA_source, nMax * (sizeof(double)));

    nMax = (long)( _msize(arrDLL_destination)/(double)(sizeof(double)) );

    // for testing do not return number of items, but their sum (as long)

    nResult = (long)sumArray(arrDLL_destination, nMax);

    free(arrDLL_destination);

    return nResult;
}
```

where `sumArray` sums up elements (an obvious function not shown here).

Using that function from Excel needs to export it and a the following declaration in VBA:

```
Declare Function receiveVBA_DLL _
  Lib "C:\_Work\MyProjects\array_excel_rw\Release\arr_xl_rw.dll" _
  Alias "receiveVBA" ( _
  ByRef arr As Double, _
  ByVal nLength As Long) As Long
```

Then we are ready to test it: for a given n the DLL function should return n*(+1)/2, which it does.

```
Sub tst_writeDLL()
Dim n As Long
Debug.Print "---"

n = 100

Debug.Print "result from DLL: " & writeDLL(n)
Debug.Print "and as VBA sum : " & CDbl(n) * (CDbl(n) + 1) / 2
End Sub
```

Note that for large n one will convert to double (in VBA) and since returns are by type `long` the result from DLL is modulo 2^32, so for n = 10^6 we get `n*(n+1)/2 = 500000500000 mod 2^32 = 1784293664.`

2. On demand of Excel read an array from C to VBA

The function wants to read data into a local arrVBA which are in a DLL function. For testing the C function to deliver its array has populated its array with 1/I and the VBA will return the sum over the elements it receives:

```
Function readDLL(ByVal nData As Long)
Dim arrVBA() As Double
Dim nResult As Long
Dim i As Long
Dim s As Double

If nData < 1 Then nData = 1

' initialize array to 0.0
ReDim arrVBA(nData)

' hand this array to DLL where it is updated by data from a local array
nResult = deliverVBA_DLL(arrVBA(1), nData)

' For testing now sum the new items (or print the array)
' and return the sum instead of the number of items
s = 0
For i = 1 To UBound(arrVBA)
  s = s + arrVBA(i)
  'Debug.Print arrVBA(i)
Next

readDLL = s
'readDLL = nResult
End Function
```

Again: note that the calling by the first array element.

The C function is

```
long __stdcall
deliverVBA(double *arrVBA_destination, long nData)
// function holding a local array which data are to be delivered to VBA
// arrVBA_destination is the array defined within
{
    double * arrDLL_source;
    long nDLL = 1000;
    long nMax;
    long i;

    arrDLL_source = (double *)calloc(nDLL,sizeof(double));

    // for testing fill with 1/i (zero based arrays in C)
    for(i=1;i<=nDLL;i++){
        arrDLL_source[i-1] = (double)1.0/i;}

    // care for size
    nMax = min( nDLL, nData );

    memmove(arrVBA_destination, arrDLL_source, nMax * (sizeof(double)));

    free(arrDLL_source);
    return nMax;
}
```

with VBA declaration

```
Declare Function deliverVBA_DLL _
  Lib "C:\_Work\MyProjects\array_excel_rw\Release\arr_xl_rw.dll" _
  Alias "deliverVBA" ( _
  ByRef arr As Double, _
  ByVal nLength As Long) As Long
```

For testing there is no simple closed form for the value, but anyway: summing the updated entries (done in `deliverVBA`) and summing 1/i gives the same result:

```
Sub tst_readDLL()
Dim n As Long
Dim i As Long
Dim check As Double
Debug.Print "---"

n = 10

Debug.Print "result from DLL: " & readDLL(n)

check = 0
For i = 1 To n
  check = check + 1 / i
Next
Debug.Print "and as VBA sum : " & check

End Sub
```

3. On demand of DLL write an array from C to VBA

Sitting within a DLL a function wants to send its data array (populated by 1/i) to Excel.
To observe it from VBA in a convenient way a trigger is used by just calling that function.

```
long __stdcall
writeVBA(l_pF_al ptr_receiveVBA, long nData)
{
    double * arrDLL_source;
    long nMax, nResult;
    long nDLL = 0;
    long i;

    arrDLL_source = (double *)calloc(nData,sizeof(double));
    nDLL = (long)( _msize(arrDLL_source)/(double)(sizeof(double)) );

    // care for size
    nMax = min( nDLL, nData );

    // for testing fill the array with 1/i (zero based arrays in C)
    for(i=1;i<=nMax;i++){
        arrDLL_source[i-1] = (double)1.0/i;}

    // call the VBA function to receive the array
    nResult = ptr_receiveVBA(arrDLL_source, nMax);

    free(arrDLL_source);
    return nResult;
}
```

```
long __stdcall
trigger_writeVBA(l_pF_al ptr_receiveVBA, long nData)
{
    return writeVBA(ptr_receiveVBA, nData);
}
```

That trigger must be declared in VBA:

```
Declare Function trigger_writeVBA_DLL _
  Lib "C:\_Work\MyProjects\array_excel_rw\Release\arr_xl_rw.dll" _
  Alias "trigger_writeVBA" ( _
  ByVal adr As Long, _
  ByVal nLength As Long) As Long
```

Of course the function needs to know its destination.

But how does the DLL know which VBA function to use?

One uses the VBA operator **`AddressOf`** to provide the DLL with Excel information, it is a pointer to the Excel function for a callback and must be wrapped:

```
Function dummyLong(ByVal x As Long) As Long
' only use: casting the return value of addressOf to a long integer
dummyLong = x
End Function
```

Within the test procedure its usage is shown.

This is applied to the VBA function `receiveDLL` (providing locally an array to receive data), do not call it in Excel directly. For testing it will return the sum of the received array:

```
Function receiveDLL( _
  ByRef arrDLL_source As Double, _
  ByVal nData As Long) As Long
' has a local array arrVBA_destination to which a DLL function wants
' to write its data
' input:  number of items to be written from DLL
' output: number of items received
Dim arrVBA_destination() As Double
Dim i As Long
Dim s As Double

If nData < 1 Then nData = 1
ReDim arrVBA_destination(nData)

' p = arr or p(i) = arr(i) does NOT work
Call RtlMoveMemory( _
  arrVBA_destination(1), _
  arrDLL_source, _
  nData * Len(arrVBA_destination(1)))

' For testing now sum the new items (or print the array)
s = 0
For i = 1 To UBound(arrVBA_destination)
  s = s + arrVBA_destination(i)
  'Debug.Print arrVBA(i)
Next
Debug.Print "sum(arrVBA_destin): " & s

receiveDLL = nData
End Function
```

This makes use of a system call to an API:

Note the calling convention for the source array both within VBA and C, in VBA one uses just 1 element (the array base), but the memory space is provided by the DLL.

```
Declare Sub RtlMoveMemory Lib "kernel32" ( _
  hpvDest As Any, _
  hpvSource As Any, _
  ByVal cbCopy As Long)
```

This is a very fast implementation of the C library function memcpy within the Windows system file kernel32.dll.

Now test whether everything fits together:

```
Sub tst_receiveDLL()
Dim m As Long
Dim adr As Long
Dim result
Dim check As Double
Dim i As Long
Debug.Print "---"

m = 1000

adr = dummyLong(AddressOf receiveDLL)
result = trigger_writeVBA_DLL(adr, m)

' check in VBA
check = 0
For i = 1 To m
  check = check + 1 / i
Next
Debug.Print "and as VBA sum    : " & check

End Sub
```

So what is happening on executing that test procedure?

It sends the address of the receiving VBA function `receiveDLL` to the C function `writeVBA` (using the trigger to access it from outside).

Then `writeVBA` uses that function pointer for a call back to Excel and sends its local array by reference - which is the first array element.

Knowing an array length (and the element size in bytes) the whole array can be accessed - this is done in the called VBA function `receiveDLL`.

Since VBA does not know a construct "pointer of type double" a system call is used to achieve it.

4. On demand of DLL read an array from VBA to C

This is just the other way round and uses the same principle: the following C function wants to get data from VBA:

```
long __stdcall
readVBA(l_pF_al ptr_deliverVBA, long nData)
{
    double * arrDLL_destination;
    long nMax, nResult;
    long nDLL = 0;

    arrDLL_destination = (double *)calloc(nData,sizeof(double));
    nDLL = (long)( _msize(arrDLL_destination)/(double)(sizeof(double)) );

    nMax = min( nDLL, nData );

    nResult = ptr_deliverVBA(arrDLL_destination, nMax);

    // for testing sum up the entries
    nResult = (long)sumArray(arrDLL_destination, nMax);

    free(arrDLL_destination);
    return nResult;
}
```

```
long __stdcall
trigger_readVBA(l_pF_al ptr_deliverVBA, long nData)
{
    return readVBA(ptr_deliverVBA, nData);
}
```

For testing the C function returns the sum of received elements.

The VBA part holding the data is

```
Function deliverDLL( _
  ByRef arrDLL_destination As Double, _
  ByVal nData As Long) As Long
' has a local array arrVBA_destination from which a DLL function wants
' to read its data
' input:  number of items to be read by DLL
' output: number of items read

Dim i As Long, iDummy As Long
Dim arrVBA_source() As Double
Dim dummy

If nData < 1 Then nData = 1
ReDim arrVBA_source(nData)

' for testing fill it with ascending natural numbers
For i = 1 To UBound(arrVBA_source)
  arrVBA_source(i) = i
Next i

Call RtlMoveMemory( _
  arrDLL_destination, _
  arrVBA_source(1), _
  nData * Len(arrDLL_destination))

deliverDLL = nData
End Function
```

Test that stuff:

```
Sub tst_deliverDLL()
Dim m As Long
Dim adr As Long
Dim result
Debug.Print "---"

m = 10

adr = dummyLong(AddressOf deliverDLL)
result = trigger_readVBA_DLL(adr, m)

Debug.Print "result in DLL  : " & result
Debug.Print "and as VBA sum : " & CDbl(m) * (CDbl(m) + 1) / 2

End Sub
```

It sends the address of the delivering VBA function `deliverDLL` to the C function `readVBA` (using the trigger to access it from outside).

Then `readVBA` uses that function pointer for a call back to Excel and sends its local array (which should receive the data) by reference - which is the first array element.

Knowing an array length (and the element size in bytes) the whole array can be accessed - this is done in the called VBA function `deliverDLL` by using an API call.