Graphica Software

160 Queen St
5th Floor
Melbourne, 3000
Victoria, Australia.
e-mail: sales@melb.graphica.com.au

Developer Note - Using SAFEARRAY

Document: safearr.doc
Issue No: Draft
Printed: 05/10/97 10:59

Table of Contents

List of Figures

List of Tables

Error! No table of contents entries found.

1. Preface

The following note provides information on using SAFEARRAY OLE Automation data types in ActiveX controls written in C++ using the "Active Template Library". In particular it discusses how SAFEARRAY data can be passed in and out of Visual Basic (Version 5).

The information was "discovered" in the creation of a communications ActiveX control that had methods which uses SAFEARRAY data type to pass "blobs" of data over the wire. This involved creating controls with Visual Basic and then looking at the created type libraries, passing data to and from the initial Visual Basic control and then trying to reproduce the same behavior using a C++ ATL control.

2. Key Words

SAFEARRAY OLE AUTOMATION ATL MIDL WIN32

3. Related Documents

[1] Microsoft Visual C++ Run-Time Library Reference Vol 3 - Active Template Library.
[2] Microsoft OLE Automation Programmer's Reference
[3] Microsoft Visual Basic Programmer's Guide (Version 4.0)

4. Changes History

Version 1.0 4 October, 1997 Graphica Software Pty. Ltd.

5. Introduction

According to the Microsoft documentation the SAFEARRAY data type is one of the basic supported OLE Automation data types.

For ActiveX controls which provide interfaces to communications facilities such as binary Send/Receive functions a SAFEARRAY array provides the preferred way to pass data into and extract data from the control.

However most ActiveX containers do not support the SAFEARRAY data type directly, and require that the SAFEARRAY be wrapped with a VARIANT type. This forces programmers to have to write extra code and is inefficent in its use of memory as extra allocation is required for the VARIANT data type wrapper.

For developers creating ActiveX controls using MFC there is no ability to specify SAFEARRAY method parameters. In addition to not supporting the direct use of the SAFEARRAY data type MFC controls do not easily support dual interfaces. For these reasons the Active Template Library (ATL) now provides the preferred way to provide optimized ActiveX controls.

The wizards included with Visual C++ Version 5.0 provide good support for the creation of ATL based controls.

Finally with the introduction of verion 5 Visual Basic is able to support the direct passing of SAFEARRAY data without the requirement for extra wrapping using VARIANTs, this will mean that other ActiveX containers will no doubt follow suite to ensure that provide full support for OLE Automation data types.

6. ATL SAFEARRAY Example

6.1 MIDL Definintion

The Microsoft Interface Definition Language (MIDL) source file for an ATL project provides the central driving code for any ATL control. The MIDL definition can be generated "automatically" via the wizard, by simply giving the function names and the parameters of the interface.
The wizard will then add the definitions to the MIDL source file and thegenerate the corresponding skeleton code stubs in the C++ source file.

The following example shows how to declare a SAFEARRAY of bytes using the MIDL.

```
interface ISafeEx : IDispatch
{
  [id(1), helpstring("Consume")]
    HRESULT Consume([in, out] SAFEARRAY(unsigned char) *Buf);
  [id(2), helpstring("Produce")]
    HRESULT Produce([in, out] SAFEARRAY(unsigned char) *Buf);
};
```

The Consume method is to pass in a blob and the Produce method is to return one. In both cases the parameters must be declared as "[in, out]" as otherwise they will not be compiled correctly and the Visual Basic Version 5 container will not work with the control.

6.2 Wizard Generated File Work Around

While this definition will be processed by the MIDL compiler corrrectly the stub header definitions and implementation files generated by the ATL wizard are incorrect and must be corrected as shown in the follow code fragments.
Original ATL wizard header file declaration

```
        STDMETHOD(Produce)(/*[in, out]*/ SAFEARRAY (unsigned char) *Buf);
        STDMETHOD(Consume)(/*[in, out]*/ SAFEARRAY (unsigned char) *Buf);
```

Modified header declaration

```
        STDMETHOD(Produce)(/*[in, out]*/ SAFEARRAY /* (unsigned char) */ **Buf);
        STDMETHOD(Consume)(/*[in, out]*/ SAFEARRAY /* (unsigned char) */ **Buf);
```

Original ATL wizard implemenation file stub

```
STDMETHODIMP SafeEx::Consume(SAFEARRAY ( unsigned char ) * Buf)
{
        // TODO: Add your implementation code here

        return S_OK;
}

STDMETHODIMP SafeEx::Produce(SAFEARRAY ( unsigned char ) * Buf)
{
        // TODO: Add your implementation code here

        return S_OK;
}
```

Modified implementation file stub

```
STDMETHODIMP SafeEx::Consume(SAFEARRAY /* ( unsigned char ) */ ** Buf)
{
        // TODO: Add your implementation code here

        return S_OK;
}

STDMETHODIMP SafeEx::Produce(SAFEARRAY /* ( unsigned char ) */ ** Buf)
{
        // TODO: Add your implementation code here

        return S_OK;
}
```

7. Passing Back a SAFEARRAY

Given the Visual Basic code fragments shown, how do we code the ATL C++ control to correctly pass back an array of data?

```
Public Sub GetBlob1()
  Dim abyBlob() As BYTE
  Dim iSize As Integer
  tControl.Produce abyBlob()

  iSize = UBound(abyBlob) - LBound(abyBlob)
  MsgBox ("Got: " & iSize)
End Sub

Public Sub GetBlob2()
  Dim abyBlob(30) As BYTE
  Dim iSize As Integer
  tControl.Produce abyBlob()

  iSize = UBound(abyBlob) - LBound(abyBlob)
  MsgBox ("Got: " & iSize)
End Sub
```

To behave correctly the Produce method should not replace any existing SAFEARRAY "headers", as this will result in Visual Basic using the disposed SAFEARRAY and ignoring the newly allocated one. Instead the called ATL control should dispose the data portion of the SAFEARRAY and reallocate a new data section and make the passed SAFEARRAY header point to this.

This is shown in the following diagram.

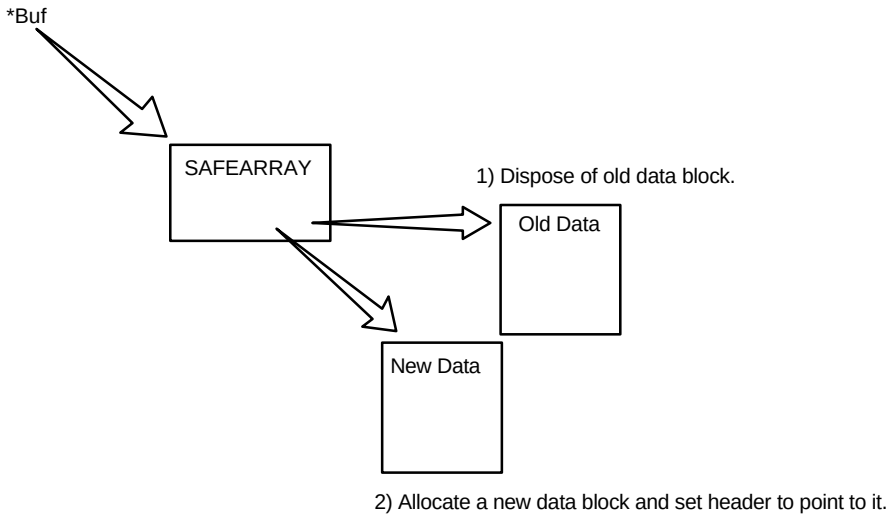

Figure 1. Maintaining existing SAFEARRAY header.

The C++ code to handle this correctly is shown in the following example (note there is no correct error handling in this code which has been check as simple as possible to illustrate the point).

```
STDMETHODIMP SafeEx::Produce(SAFEARRAY /* ( unsigned char ) */ ** tBuf)
//
// Note: Pass back a 100 Byte array, to illustrate what we do.
//
{
  HRESULT tRes = S_OK;
  SAFEARRAYBOUND tBnd[1];
  void *pvData;

  if (*tBuf) {
    //
    // Dispose of the original data block.
    //
    if (SafeArrayGetElemsize(*tBuf) == sizeof(unsigned char) &&
        SafeArrayGetDim(*tBuf) == 1) {
      SafeArrayDestroyData(*tBuf);
```

```
      (*tBuf)->rgsabound[0].lLbound = 0;
      (*tBnd)->rgsabound[0].cElements = 100;
      if (! FAILED(SafeArrayAllocData(*tBuf))) {
        if (! FAILED(SafeArrayAccessData(*tBuf, &pvData))) {
          memset(pvData, 0, 100);
          SafeArrayUnaccessData(*tBuf);
        }
      }
    }
  }
  else {
    //
    // Create a completely new Safe Array
    //
    tBnd[0].lLBound = 0;
    tBnd[0].cElements = 100;
    *tBuf = SafeArrayCreate(VT_U1, 1, tBnd);
    if (*tBuf) {
      if (! FAILED(SafeArrayAccessData(*tBuf, &pvData))) {
        memset(pvData, 0, 100);
        SafeArrayUnaccessData(*tBuf);
      }
    }
  }

  return(tRes);
}
```

This code has been trailed with Visual Basic Version 5 and found to behave the same as if the control had been written directly in Visual Basic.

End of Document