Using DCOM Wrapper Objects

This chapter covers the following topics:

- Properties
- Set_<Property>
- Get_<Property>
- Properties and Groups
- Using Generated Object Methods
- Handling Arrays
- Handling Groups
- Handling Periodic Groups

Properties

Every generated object has a set of standard properties. If there are any group definitions in the IDL file, additional properties exist to set and get attributes inside groups and to access groups. See also *Properties and Groups*.

Note:
The notation of property names is case-sensitive.

Set_<Property>

C++

```
/* Initialize C O M */
...
/* Get pointer to instance -> pDspObj */
...
/* Get DispId of property -> DispId_Property */
...

VARIANT presult;
HRESULT res = 0;
DISPID mydispid = DISPID_PROPERTYPUT;

VARIANTARG args[1];
VariantInit(args);

/* Next two lines depends on type of corresponding property */
V_R4(args) = <Value>
V_VT(args) = VT_R4;

DISPPARAMS params;
params.rgvarg = args;
params.cArgs = 1;
```

```
params.cNamedArgs = 1;
params.rgdispidNamedArgs = &mydispid;

EXCEPINFO ExcpInfo;
unsigned int pargerr;

res = pDspObj->Invoke( DispId_Property,
                       IID_NULL,
                       LOCALE_SYSTEM_DEFAULT,
                       DISPATCH_PROPERTYPUT,
                       &params,
                       &presult,
                       &ExcpInfo,
                       &pargerr
                       );

if ( FAILED( res ) )
{
   /* Exception handling */
   ...
}
```

Visual Basic

```
Dim obj As Object
Dim propValue As <Property Type>
...
Set obj = CreateObject("<Object Name>")
...
obj.<Property Name>=<Value>
...
```

Get_<Property>

C++

```
/* Initialize C O M */
...
/* Get pointer to instance -> pDspObj */
...
/* Get DispId of property -> DispId_Property */

void* psret;
HRESULT res;
DISPPARAMS params = { NULL, NULL, 0, 0 };
VARIANT result;
EXCEPINFO except;
unsigned int argerr;

res = pDspObj->Invoke(   DispId_Property,
                       IID_NULL,
                       0,
                       DISPATCH_PROPERTYGET,
                       &params,
                       &result,
                       &except,
                       &argerr
                       );

if ( SUCCEEDED( res ) )
```

```
{
    /* prepare psret */
    ...
    if ( psret != NULL )
    {
        switch ( V_VT(&result) )
        {
            case VT_BSTR:
                psret = (BSTR)(V_BSTR(&result));
                break;
            case VT_I4:
                psret = (long*)&(V_I4(&result));
                break;
            case VT_R4:
                psret = (float*)&(V_R4(&result));
                break;
            case VT_DISPATCH:
                psret = V_DISPATCH(&result);
                break;

            /* cases of other variant types*/
                ...

            default:
               /* error : unsupported variant type */
        }
    }
}
else
{
    /* Error Handling */
    ...
}
```

Visual Basic

```
Dim obj As Object
Dim propValue As <Property Type>
...
Set obj = CreateObject("<Object Name>")
...
propValue = obj.<Property Name>
...
```

Properties and Groups

Additional properties are defined for group handling. There are properties to set or get values of group members (similar to set/get on standard properties) and to get a handle for an interface representing a (sub)group or a pointer to a dispatch object representing a (sub)group.

Notes:

1. An array inside a (periodic) group is handled like a scalar attribute, i.e. it is passed as an array only by pure name (see also grammar below).
2. Program names change to uppercase, attribute and group names change to lowercase.

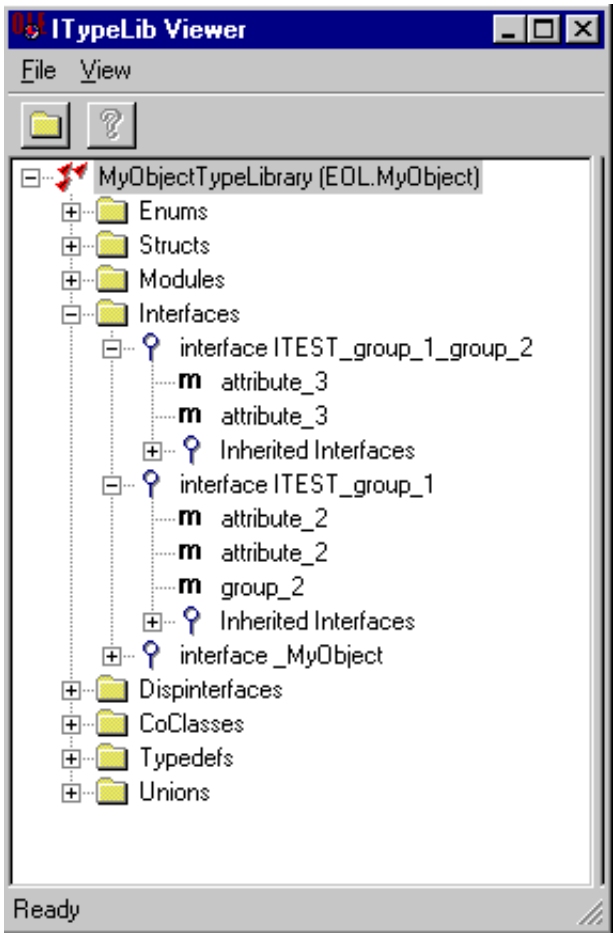
Grammar for access to group members

```
<Group Attribute> ::=     <ObjectName>.<Qualifier>
<Qualifier>       ::=     <TopLevel>.<Levels>.<Attribute>
                              | <TopLevel>.<Attribute>
                              | <Attribute>
<TopLevel>        ::=     <program name (uppercase)>_<Level>
                              | <program alias name>
<Levels>          ::=     <Level>.<Levels>
                              | <Level>
<Level>           ::=     <group name(lowercase)> <Index>
<Attribute>       ::=         "attribute name (lowercase)"
<Index>           ::=          ""
                              | "(Index_1)"
                              | "(Index_1, Index_2)"
                              | "(Index_1, Index_2, Index_3)"
```

Note:
Every index starts with 0.

IDL

```
Library ’MyObject’ Is
  Program ’TEST’ Is
    Define Data Parameter
      1 Group_1
      2   Attribute_2          (I4)
      2   Group_2
      3    Attribute_3         (I4)
    End-Define
```



C++

```
IDISPATCH *pDspObj, *pDspTest_group_1, *pDspTest_group_2;

    /*  Initialize C O M */
   ...
    /* Get pointer to instance -> pDspObj */
    ...
    /* some declarations and initializations*/
   ...
   {
       OLECHAR *Names[] = { L"TEST_group_1" };
       DISPID id[1];

       res = pDspObj->GetIDsOfNames( IID_NULL;
                                     Names,
                                     1,
                                     0,
                                     id );

       DISPPARAMS params = { NULL, NULL, 0, 0 };
       VARIANT presult;

       res = pDspObj->Invoke( id[0],
                              IID_NULL,
                              0,
                              DISPATCH_PROPERTYGET,
                              &params,
```

```
                            &presult,
                            NULL,
                            0);

        if ( V_VT( &result ) == VT_DISPATCH )
        {
            pDspTest_group_1 = V_DISPATCH(&result);
        }
        else
        {
            /* error handling */
        }
    }
    {
         OLECHAR *Names[] = { L"attribute_2", L"group_2" };
         DISPID id[2];

         res = pDspTest_group_1->GetIDsOfNames( IID_NULL,
                                                Names,
                                                2,
                                                0,
                                                id );

         DISPPARAMS params = { NULL, NULL, 0, 0 };
         VARIANT presult;

         res = pDspTest_group_1->Invoke( id[0],
                                         IID_NULL,
                                         0,
                                         DISPATCH_PROPERTYGET,
                                         &params,
                                         &presult,
                                         NULL,
                                         0);

         if  ( V_VT( &result ) == VT_I4 )
         {
             /* plong2 = pointer to a long */
             plong2 = (long*)&(V_I4(&result));
         }
         else
         {
            /* error handling */
         }

         res = pDspTest_group_1->Invoke( id[1],
                                         IID_NULL,
                                         0,
                                         DISPATCH_PROPERTYGET,
                                         &params,
                                         &presult,
                                         NULL,
                                         0);

         if  ( V_VT( &result ) == VT_DISPATCH )
         {
             pDspTest_group_2 = V_DISPATCH(&result);
         }
         else
         {
             /* error handling */
         }
```

```
    }
    {
        OLECHAR *Names[] = { L"attribute_3" };
        DISPID id[1];

        res = pDspObj->GetIDsOfNames( IID_NULL,
                                      Names,
                                      1,
                                      0,
                                      id );

        DISPPARAMS params = { NULL, NULL, 0, 0 };
        VARIANT presult;

        res = pDspTest_group_2->Invoke( id[0],
                                        IID_NULL,
                                        0,
                                        DISPATCH_PROPERTYGET,
                                        &params,
                                        &presult,
                                        NULL,
                                        0);

        if  ( V_VT( &result ) == VT_I4 )
        {
           /* plong3 = pointer to a long */
           plong3 = (long*)&(V_I4(&result));
        }
        else
        {
            /* error handling */
        }
    }
```

Visual Basic

```
Dim obj As Object
Dim lValue As Long
...
Set obj = CreateObject("<Object Name>")
...
’ Set value from attribute inside a group
obj.TEST_group_1.attribute_2 = lValue
obj.TEST_group_1.group_2.attribute_3 = lValue
...
’ Get value from attribute inside a group
lValue = obj.TEST_group_1.attribute_2
lValue = obj.TEST_group_1.group_2.attribute_3
...
```

IDL

```
Program ’TEST’ Is
  Define Data Parameter
    1  Group_1A
    2    Attribute_2       (I4)
    2    Group_2A
    3      Attribute_3     (I4)
    2    Group_2B  (/3,4)
    3      Attribute_4     (I4)
    1  Group_1B    (/5)
    2    Group_2C  (/3,4)
```

```
    3      Attribute_5     (I4)
    2    Group_2D
    3      Attribute_6     (I4)
    3      Group_3
    4        Attribute_7 (I4)
  End-Define
```

C++

```
/* Example: access to group_1b(2).group_2c(3 , 4).attribute5 */
Idispatch *pdspObj, *pdspPTEST_group_1b, *pdspPTEST_group_1b_group_2c;
/* get dispatch pointer to object -> pdspObj */
...
/* get_TEST_group_1b(2) */
{
    OLECHAR *Names[] = { L"TEST_group_1b" };
    DISPID id[1];

    res = pdspObj->GetIDsOfNames( IID_NULL,
                                  Names,
                                  1,
                                  0,
                                  id );
    VARIANTARG args[1];
    VariantInit(args);
    V_I4(args) = 1;
    V_VT(args) = VT_I4;

    DISPPARAMS params;
    params.rgvarg = args;
    params.cArgs = 1;
    params.cNamedArgs = 0;
    VARIANT presult;
    res = pdspObj->Invoke( id[0],
                           IID_NULL,
                           0,
                           DISPATCH_PROPERTYGET,
                           &params,
                           &presult,
                           NULL,
                           0);
    if ( V_VT( &result ) == VT_DISPATCH )
    {
        pdspTest_group_1b = V_DISPATCH(&result);
    }
    else
    {
        /* error handling */
    }
}
/* get_group_2c(2,3) */
{
    OLECHAR *Names[] = { L"group_2c" };
    DISPID id[1];

    res = pdspTest_group_1b ->GetIDsOfNames( IID_NULL,
                                             Names,
                                             1,
                                             0,
                                             id );
    VARIANTARG args[2];
    VariantInit(args);
```

```
    VariantInit(args+1);
    V_I4(args+1) = 1;
    V_VT(args+1) = VT_I4;
    V_I4(args) = 2;
    V_VT(args) = VT_I4;

    DISPPARAMS params;
    params.rgvarg = args;
    params.cArgs = 2;
    params.cNamedArgs = 0;
    VARIANT presult;
    res = pdspTest_group_1b ->Invoke( id[0],
                                      IID_NULL,
                                      0,
                                      DISPATCH_PROPERTYGET,
                                      &params,
                                      &presult,
                                      NULL,
                                      0);
    if ( V_VT( &result ) == VT_DISPATCH )
    {
        pdspTest_group_1b_group_2c = V_DISPATCH(&result);
    }
    else
    {
        /* error handling */
    }
}
/* get attribute5 */
long ltemp;
{
    OLECHAR *Names[] = { L"attribute5" };
    DISPID id[1];

    res = pdspTest_group_1b_group_2c ->GetIDsOfNames( IID_NULL,
                                                      Names,
                                                      1,
                                                      0,
                                                      id );
    DISPPARAMS params = { NULL, NULL, 0, 0 };
    VARIANT presult;
    res = pdspTest_group_1b_group_2c ->Invoke( id[0],
                                               IID_NULL,
                                               0,
                                               DISPATCH_PROPERTYGET,
                                               &params,
                                               &presult,
                                               NULL,
                                               0);
    if ( V_VT( &result ) == VT_I4)
    {
        ltemp = V_I4(&result);
    }
    else
    {
        /* error handling */
    }
}
```

Visual Basic

```
Dim obj As Object
Dim lValue As Long
Dim l0, l1, l2 As Long
...

Set obj = CreateObject("<Object Name>")
...
' Set value from attribute inside a group
obj.TEST_group_1a.attribute_2 = lValue
obj.TEST_group_1a.group_2a.attribute_3 = lValue
obj.TEST_group_1a.group_2b( i1 , i2 ).attribute_4 = lValue
...
obj.TEST_group_1b( i0 ).group_2c( i1 , i2 ).attribute_5 = lValue
obj.TEST_group_1b( i0 ).group_2d.attribute_6 = lValue
obj.TEST_group_1b( i0 ).group_2d.group_3.attribute_7 = lValue
...
' Get value from attribute inside a group
...
lValue = obj.TEST_group_1a.attribute_2
lValue = obj.TEST_group_1a.group_2a.attribute_3
lValue = obj.TEST_group_1a.group_2b( i1 , i2 ).attribute_4
lValue = obj.TEST_group_1b( i0 ).group_2c( i1 , i2 ).attribute_5
lValue = obj.TEST_group_1b( i0 ).group_2d.attribute_6
lValue = obj.TEST_group_1b( i0 ).group_2d.group_3.attribute_7
...
```

Using Generated Object Methods

Calling Remote Procedures as Functions

The IDL syntax allows you to define (remote) procedures only. This is similar to Natural, which knows only procedures (referred to in Natural as subprograms). Neither IDL nor Natural have the concept of a function. A function is a procedure which, in addition to the parameters, returns some value.

It is possible to treat the OUT parameter of a procedure as the return value of a function. Using this technique, a procedure can be used as a function. The DCOM Wrapper generates a function rather than a procedure when the following two conditions are met:

- the last parameter of the procedure definition is of type OUT;
- this last parameter of the procedure definition has the name `Function_Result`.

As an example, see the *calc.idl* file in the subdirectory *Examples\DCOM Wrapper\calc* of the EntireX installation:

```
Program 'Calc' Is
  Define Data Parameter
   1 Operator_         (A1) In
   1 Operand_1         (I4) In
   1 Operand_2         (I4) In
   1 Function_Result   (I4) Out
  End-Define
```

From the above specification, the DCOM Wrapper generates an object that can be called from Visual Basic as follows:

```
Dim result As Long
. . .
result = CALCOLEObj.calc (’+’, 1234, 1234)
```

If the last parameter had a name other than `Function_Result` in the IDL file, the call in Visual Basic would look as follows:

```
CALCOLEObj.calc ’+’, 1234, 1234, result
```

Handling Arrays

ActiveX automation technology supports only a restricted set of data types (see supported data types).

The DCOM Wrapper supports the use of arrays with up to three dimensions. For all base types, the DCOM Wrapper uses the so-called SAFEARRAY data type for mapping arrays to ActiveX data types.

Note:
When using a SAFEARRAY it is expected that the array has the right size. It is not recommended to leave any member of an array undefined.

[in] Parameters

IDL

```
Program ’CPROG’; is
  Define data parameter
   1 IVALUE       (I4)            IN
   1 IARRAY       (A80/1:9)       IN
  End-define
```

Visual Basic

```
...
dim arr( )
redim arr(8)
...
for n = 0 to 8
    arr(n) = "ONLY IN " & (n+1)
    document.write arr(n)
next
WrapperObject.CPROG 123, arr
```

C/C++

```
...
long i4_single = 12345678;
SAFEARRAY *iarray;
SAFEARRAYBOUND rgsabound_dim1[] = {9, 0};

char temp [32];
OLECHAR wtemp[32];
iarray = SafeArrayCreate(VT_BSTR, 1, rgsabound_1 );
for (i = 0; i <9; i++)
{
    sprintf (temp,"I%d",(i+1)*1);
```

```
    mbstowcs(wtemp, temp, 80);
    iarray[i] = SysAllocString(wtemp);
}

VARIANTARG args[2];
VariantInit(args);
VariantInit(args+1);
DISPPARAMS params;
V_ARRAYREF(args+1) = & iarray;
V_VT(args+1) = VT_ARRAY|VT_BSTR|VT_BYREF;
V_I4REF(args) = &i4_single;
V_VT(args) = VT_I4|VT_BYREF;
params.rgvarg = args;
params.rgdispidNamedArgs = cNames > 1 ? id+1 : 0;
params.cArgs = 2;
params.cNamedArgs = cNames-1 ;
EXCEPINFO pExcpInfo;
res = pDspObj->Invoke( id[0],
                       IID_NULL,
                       0,
                       DISPATCH_METHOD,
                       &params,
                       NULL,
                       &pExcpInfo,
                       0 );
...
```

[InOut] Parameters

IDL

```
...
Program ’APROG’; is
  Define data parameter
    1 IVALUE (I4) IN
    1 IOARRAY (A80/1:10) IN OUT
  End-define
```

Visual Basic

```
...
Dim arr( )
ReDim arr(9)
...
for n = 0 to 9
    arr(n) = "IN OUT" & (n+1)
    document.write arr(n)
next
WrapperObject.APROG 123, arr
For each strval in arr
    Document.write strval
Next
```

C/C++

```
...
long i4_single = 12345678;
SAFEARRAY *ioarray;
SAFEARRAYBOUND rgsabound_dim1[] = {10, 0};

char temp [32];
```

```
OLECHAR wtemp[32];
ioarray = SafeArrayCreate(VT_BSTR, 1, rgsabound_1 );
for (i = 0; i <10; i++)
{
    sprintf (temp,"I%d",(i+1)*1);
    mbstowcs(wtemp, temp, 80);
    ioarray[i] = SysAllocString(wtemp);
}

VARIANTARG args[2];
VariantInit(args);
VariantInit(args+1);
DISPPARAMS params;
V_ARRAYREF(args+1) = &ioarray;
V_VT(args+1) = VT_ARRAY|VT_BSTR|VT_BYREF;
V_I4REF(args) = &i4_single;
V_VT(args) = VT_I4|VT_BYREF;
params.rgvarg = args;
params.rgdispidNamedArgs = cNames < 1 ? id+1 : 0;
params.cArgs = 2;
params.cNamedArgs = cNames-1 ;
EXCEPINFO pExcpInfo;
res = pDspObj->Invoke( id[0],
                       IID_NULL,
                       0,
                       DISPATCH_METHOD,
                       &params,
                       NULL,
                       &pExcpInfo,
                       0 );
...
```

[Out] Parameters

IDL

```
Program 'BPROG'; is
  Define data parameter
    1 IVALUE (I4) IN
    1 OARRAY (A80/1:10) OUT
  End-define
```

Visual Basic

```
...
Dim OARRAY() ReDim OARRAY(10) 'see Notes on Visual Basic
...
WrapperObject.BPROG 123, arr
For each strval in arr
    Document.write strval
Next
```

C/C++

In C/C++ INOUT and OUT parameters are handled in same way. See the example for INOUT parameters.

Visual Basic Notes:

Visual Basic arrays start with index 0. VBScript does not support the "dim myarray(... to ...)" notation. Because array sizes are checked you have to dimension your array *N*-1 when it contains *N* elements. INOUT arrays have to be defined like OUT arrays and be redimensioned to the required size.

Note:
The DCOM Wrapper can create objects that support VARIANT references. Scripting languages such as VBScript pass output parameters by VARIANT references and not via an exactly defined type. For example, when a method of a COM interface has an out parameter of type String, Visual Basic passes a reference to a VARIANT to get the out parameter. DCOM Wrapper objects try to convert these references into the required reference type.

Handling Groups

Referencing attribute values in groups is demonstrated in *Properties and Groups*. After you have set the attribute values, the method call must be prepared.

The argument list of the method call consists of all attributes and groups at level 1 (that is, attributes in lines beginning with 1 in the IDL file). The group is represented by a dispatch pointer (C/C++) or interface (Visual Basic) that must be passed as an argument. A special property has been introduced to get this group argument (see *Properties and Groups*).

Example

```
Program 'EXAMPLE1' Is
  Define Data Parameter
    1 MyStruct
    2  MyStruct1
    3    MyLong(I4)
    3    MyFloat (F4)
    2  MyLong (I4)
    2  MyFloat (F4)
    1 MyStructAsString (A253)
    1 Function_Result (I4) Out
  End-Define
```

C/C++

```
//
// initialize structure values
//
IDispatch *pDspObj, *pDspEXAMPLE1_mystruct, *pDspEXAMPLE1_mystruct_mystruct1;
DISPID functionID;
...
/* get pointer to group mystruct in program Example1 */
{
    /* get ID of group mystruct and program Example1 */
    OLECHAR *Names[] = { L"EXAMPLE1_mystruct", L"EXAMPLE1" };
    DISPID id[2];

    res = pDspObj->GetIDsOfNames( IID_NULL,
                                  Names,
                                  2,
                                  0,
                                  id );
```

```
    /* store function id */
    functionID = id[1];
    DISPPARAMS params = { NULL, NULL, 0, 0 };
    VARIANT presult;
    res = pDspObj->Invoke( id[0],
                           IID_NULL,
                           0,
                           DISPATCH_PROPERTYGET,
                           &params,
                           &presult,
                           NULL,
                           0);
    if ( V_VT( &result ) == VT_DISPATCH )
    {
        pDspEXAMPLE1_mystruct = V_DISPATCH(&result);
    }
    else
    {
        /* error handling */
    }
}
{
    OLECHAR *Names[] = { L"mystruct_1" };
    DISPID id[1];

    res = pDspEXAMPLE1_mystruct ->GetIDsOfNames( IID_NULL,
                                                 Names,
                                                 1,
                                                 0,
                                                 id );
    DISPPARAMS params = { NULL, NULL, 0, 0 };
    VARIANT presult;
    res = pDspEXAMPLE1_mystruct ->Invoke( id[0],
                                          IID_NULL,
                                          0,
                                          DISPATCH_PROPERTYGET,
                                          &params,
                                          &presult,
                                          NULL,
                                          0);
    if ( V_VT( &result ) == VT_DISPATCH )
    {
        pDspEXAMPLE1_mystruct_mystruct1 = V_DISPATCH(&result);
    }
    else
    {
        /* error handling */
    }
}
...
/* call the function EXAMPLE1 */
{
    BSTR MyString = SysAllocString(L"");
    VARIANTARG args[2];
    VariantInit(args);
    VariantInit(args+1);
    V_DISPATCH(args+1) = pDspEXAMPLE1_mystruct;
    V_VT(args+1) = VT_DISPATCH;
    V_BSTRREF(args) = &MyString;
    V_VT(args) = VT_BSTR|VT_BYREF;
    DISPPARAMS params;
    params.rgvarg = args;
```

```
    params.cArgs = 2;
    params.cNamedArgs = 0;
    EXCEPINFO ExcpInfo;
    unsigned int uArgErr;
    /* call procedure EXAMPLE1 */
    res = pDspObj->Invoke( functionID,
                           IID_NULL,
                           LOCALE_SYSTEM_DEFAULT,
                           DISPATCH_METHOD,
                           &params,
                           &presult,
                           &ExcpInfo,
                           &uArgErr );
    if (FAILED(res))
    {
        /* error handling */
    }
    if ( V_VT(&result) == VT_I4 )
    {
        ltemp = V_I4(&result);
    }
    else
    {
        /* Wrong data type */
    }
}
```

Visual Basic

```
Dim obj As Object
...
Set obj = CreateObject("<object name>")
...
Dim retString As String
Dim retCode As Long

retString = " "
obj.EXAMPLE1_mystruct.mystruct1.mylong = ...
obj.EXAMPLE1_mystruct.mystruct1.myfloat = ...
obj.EXAMPLE1_mystruct.mylong = ...
obj.EXAMPLE1_mystruct.myfloat = ...


obj.EXAMPLE1obj.EXAMPLE1_mystruct, retString

’ get value in structure
myVar1 = obj.EXAMPLE1_mystruct.mystruct1.mylong
myVar2 = obj.EXAMPLE1_mystruct.mylong
```

Handling Periodic Groups

Referencing attribute values in periodic groups is demonstrated in *Properties and Groups*. The argument list of the method call consists of all attributes and (periodic) groups at level 1 (that is, attributes in lines beginning with 1 in the IDL file). The group is represented by a dispatch pointer (C/C++) or interface (Visual Basic) that must be passed as an argument. If a periodic group at level 1 exists, it is passed as an array of dispatch pointers (C/C++) or interfaces (Visual Basic) in the argument list (see *Handling Arrays*). A special property (<*PROGRAM NAME*>_<*group name*>*_all*) has been introduced to get this array of group arguments.

Example

```
Program 'PGROUP' Is
  Define Data Parameter
    1 TABLE (/5,4) 2 CELL (F8)
    1 RESULTH (F8/4)
    1 RESULTV (F8/5)
  End-Define
```

C/C++

```
HRESULT res, hr;
VARIANT result;
long lindex[2];

/* safearray init */
SAFEARRAY* psaDsp;
SAFEARRAYBOUND boundaries[2];
boundaries[0].lLbound=0;
boundaries[0].cElements= 5;
boundaries[1].lLbound=0;
boundaries[1].cElements= 4;

SAFEARRAY* psaResH;
SAFEARRAYBOUND boundariesH[1];
boundariesH[0].lLbound=0;
boundariesH[0].cElements= 4;

SAFEARRAY* psaResV;
SAFEARRAYBOUND boundariesV[1];
boundariesV[0].lLbound=0;
boundariesV[0].cElements= 5;

psaDsp = SafeArrayCreate(VT_DISPATCH, 2, boundaries);
psaResH = SafeArrayCreate(VT_R8, 1, boundariesH);
psaResV = SafeArrayCreate(VT_R8, 1, boundariesV);
...
OLECHAR *Names[] = { L"PGROUP", L"PGROUP_table" };
DISPID id[2];
pDspObj -> GetIDsOfNames( IID_NULL,
                          Names,
                          2,
                          0,
                          id );

DISPID FuncId_PGROUP = id[0];
DISPID StructId_PGROUP_table = id[1];
IDispatch* dspTemp2;

/* get each dispatch pointer to elements of periodic group */
for ( lindex[0] = 0; lindex[0] < 5; lindex[0]++)
{
    for ( lindex[1] = 0; lindex[1] < 4; lindex[1]++)
    {
        VARIANTARG args[2];
        VariantInit(args);
        VariantInit(args+1);
        V_I4(args+1) = lindex[0];
        V_VT(args+1) = VT_I4;
        V_I4(args) = lindex[1];
        V_VT(args) = VT_I4;
        DISPPARAMS params;
```

```
        params.rgvarg = args;
        params.cArgs = 2;
        params.cNamedArgs = 0;
        res = pDspObj->Invoke( StructId_PGROUP_table,
                               IID_NULL,
                               0,
                               DISPATCH_PROPERTYGET,
                               &params,
                               &result,
                               NULL,
                               NULL );

        if ( SUCCEEDED( res ) || ( V_VT(&result) == VT_DISPATCH ))
        {
            IDispatch* dspTemp = V_DISPATCH(&result);
            dspTemp2 = dspTemp;
            SafeArrayPutElement(psaDsp, (long*) lindex, dspTemp)
        }
    }
}
/* get dispid for setting values */
OLECHAR *Names[] = { L"cell" };
DISPID id[1];
pDspObj -> GetIDsOfNames( IID_NULL,
                          Names,
                          1,
                          0,
                          id );
DISPID dispidTableCell = id[0];
/* set values */
DISPID mydispid = DISPID_PROPERTYPUT;
for ( lindex[0] = 0; lindex[0] <5; lindex[0]++)
{
   for ( lindex[1] = 0; lindex[1] < 4; lindex[1]++)
   {
        hr = SafeArrayGetElement(psaDsp, (long*) lindex, &dspTemp2)
        VARIANTARG args[1];
        VariantInit(args);
        V_R8(args) = <double value>;
        V_VT(args) = VT_R8;
        DISPPARAMS params;
        params.rgvarg = args;
        params.cArgs = 1;
        params.cNamedArgs = 1;
        params.rgdispidNamedArgs = &mydispid;
        res = dspTemp2->Invoke( dispidTableCell,
                                IID_NULL,
                                LOCALE_SYSTEM_DEFAULT,
                                DISPATCH_PROPERTYPUT,
                                &params,
                                &result,
                                NULL,
                                NULL );
    }
}
/* call function */
VARIANTARG args[3];
VariantInit(args+2);
VariantInit(args+1);
VariantInit(args);
V_ARRAYREF(args) = &psaResV;
V_VT(args) = VT_R8|VT_ARRAY|VT_BYREF;
```

```
V_ARRAYREF(args+1) = &psaResH;
V_VT(args+1) = VT_R8|VT_ARRAY|VT_BYREF;
V_ARRAYREF(args+2) = &psaDsp;
V_VT(args+2) = VT_DISPATCH|VT_ARRAY|VT_BYREF;
DISPPARAMS params;
params.rgvarg = args;
params.cArgs = 3;
params.cNamedArgs = 0;
res = pDspObj->Invoke( FuncId_PGROUP,
                       IID_NULL,
                       LOCALE_SYSTEM_DEFAULT,
                       DISPATCH_METHOD,
                       &params,
                       &result,
                       NULL,
                       NULL );
...
```

Visual Basic

```
’ declare array of interface pointer for structures access
Dim i1, i2 As Long
’ define as dynamic arrays
Dim dblhres() As Double
Dim dblvres() As Double
’ set safearray bounds
ReDim dblhres(0 To 3)
ReDim dblvres(0 To 4)
’ Initialize values
...
’ set values in structure
For i1 = 0 To 4
    For i2 = 0 To 3
        obj.PGROUP_table(i1, i2).cell = ...
    Next i2
Next i1
’ method call
obj.PGROUP obj.PGROUP_table_all, dblhres, dblvres
...
End Sub
```