Using the DCOM Wrapper with LotusScript

Programming with LotusScript (© Lotus Corp.) is similar to programming with Visual Basic. In some cases, however, there are differences which are described in the following examples. This section refers to LotusScript version 4.0. Technical changes and enhancements of other LotusScript versions are not considered here. Please refer to the LotusScript manuals.

This chapter covers the following topics:

- Creating and Destroying Objects
- Working with Arrays
- Working with Group(s)
- Working with Arrays of Groups

Creating and Destroying Objects

LotusScript does not support the data type Object. An object must be wrapped with the data type Variant.

IDL Example

```
Library 'LotusScript':'LotusScript' is
program 'Calc':'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
```

LotusScript Example

```
Sub calc
' define variant to hold the object reference
Dim myObject As Variant
' Create an instance of DCOM Wrapper object dll
Set myObject = CreateObject("LotusScript")
result = myObject.Calc ( "+", index, index )
' Do not forget to release the object
Set myObject = Nothing
End Sub
```

Working with Arrays

When you are using an array as a return value (OUT or INOUT parameter), wrap the array with the data type Variant, otherwise the returning array is not set.

IDL Example

```
Library ’LotusScript’:’LotusScript’ is
Program ’longArray’:’longArray’ is
Define Data Parameter
1 FourByteIntegerArrayIn (I4/4) In
1 FourByteIntegerArrayOut (I4/4) Out
1 FourByteIntegerArrayInOut (I4/4) In Out
End-Define
```

LotusScript Example

```
Sub longArray
Dim myObject As Variant
Dim index As Long
Dim inLongArray() As Long
Dim outLongArray() As Long
Dim inoutLongArray() As Long
Redim inLongArray(3)
Redim outLongArray(3)
Redim inoutLongArray(3)
Dim inVariant As Variant
Dim outVariant As Variant
Dim inoutVariant As Variant
’ Create an instance of DCOM Wrapper object dll
Set myObject = CreateObject("LotusScript")
’ fill arrays with data
For index= 0 To 3
inLongArray(index) = (index + 1) * 1111
inoutLongArray(index)= (index + 1) * (-2222)
Next index
’ wrap arrays in variants
inVariant = inLongArray
outVariant = outLongArray
inoutVariant = inoutLongArray
Call myObject.longArray (inVariant, outVariant, inoutVariant)
Set myObject = Nothing
End Sub
```

Working with Group(s)

When you are using simple groups, the same differences to using Visual Basic that were mentioned above also apply. When you assign a group object to a variable in your script, use the data type Variant and do not forget to release it at the end of the program. When you are using an array of groups, it is necessary to create an array of data type Variant and to assign each group object to this array. Do not forget to release objects in the array at the end of the program.

Working with a Simple Group

```
Software AG IDL Example:
Library ’LotusScript’:’LotusScript’ is
Program ’SimpleGroup’ is
Define Data Parameter
1 Group1
2 Group2
3 MyLong (I4)
3 MyFloat (F4)
2 MyLong (I4)
```

```
2 MyFloat (F4)
1 MyStructAsString (A253)
1 Function_Result (I4) Out
End-Define
```

LotusScript Example

```
Sub prog1
Dim number As Long
Dim name As String
Dim result As Long
Dim myObject As Variant
’ Create an instance of DCOM Wrapper object dll
Set myObject = CreateObject("LotusScript")
’ Set value
myObject.SimpleGroup_group1.MyLong = 123
myObject.SimpleGroup_group1.group2.MyLong = 1233
myLong = myObject.SimpleGroup_mystruct.mystruct1.MyLong
myLong = myObject.SimpleGroup(myObject.SimpleGroup_mystruct,name)
’ Release object
Set myObject = Nothing
End Sub
```

Working with Arrays of Groups

IDL Example

```
Library ’LotusScript’:’LotusScript’ is
Program ’AddNumbers’:’AddNumbers’ is
Define Data Parameter
1 List (/9)
2 Number (F8)
1 RESULT (F8)
End-Define
```

LotusScript Example

```
Sub prog2
Dim myObject As Variant
Dim sum As Double
Dim index As Long
’ storage for group objects

Dim listObj(9) As Variant
Set myObject = CreateObject("LotusScript")
For index = 0 To 9
Set listObj(index) = exxObj.AddNumbers_List(index)
’ set value
listObj(index).cell = 1000
Next index
’ method call
myObject.AddNumbers listObj, sum
’ Release group object
For index = 0 To 9
Set retObj(index) = Nothing
Next index
’ Release object
Set myObject = Nothing
End Sub
```