Generated DCOM Wrapper Objects

This chapter covers the following topics:

- Supported Data Types
- Code Generation Process
- Location of DCOM Wrapper Objects
- Standard Wrapper Properties
- Handling Complex Data Types
- Calling Remote Procedures as Functions
- Standard Wrapper Methods
- Shared C Runtime Environment
- Registering a Wrapper Object
- Deployment of Wrapper Objects
- Using Wrapper Objects with DCOM

Supported Data Types

All the Software AG IDL data types are mapped to the COM data types as shown in the table below.

IDL Data Type	C++/COM Data Type	Visual Basic Data Type	Note
A	BSTR	String	See note below.
AV	BSTR	String	See note below.
B	unsigned char	Byte	See note below.
BV	SAFEARRAY(unsigned char)	Byte	See note below.
D	DATE	DATE	
F4	float	Single	
F8	double	Double	
I1	short	Integer	Range is -128 to 127.
I2	short	Integer	
I4	long	Long	
K	BSTR	String	See note below.
KV	BSTR	String	See note below.
L	VARIANT_BOOL	Boolean	
N	BSTR/double	String/Double	Depending on code generation option Code Type.
P	BSTR/double	String/Double	Depending on code generation option Code Type.
T	DATE	DATE	

Note:
The maximum length you can specify depends on your hardware configuration and your software environment apart from EntireX. There is, however, an absolute limit (1 GB) that cannot be exceeded.

Code Generation Process

The DCOM Wrapper is used to generate an ActiveX automation server from an IDL file. The DCOM Wrapper Wizard generates a batch (script) file in the directory where the IDL file resides:

- *<idl-filename>.bat* on Windows (see *Platform Coverage*)

This file uses the Software AG IDL Compiler and appropriate C++ development tools to generate the ActiveX automation server code. See *Software AG IDL File* for IDL file specifications.

Location of DCOM Wrapper Objects

The locations of generated Wrapper objects depend on the directory containing the corresponding IDL file. The names of generated Wrapper objects depend on the names of the libraries defined in the IDL file.

Wrapper automation objects may be renamed using the **Naming Options** dialog, but the names of the generated Wrapper files remain unaffected.

The automation object name is used to define the COM program and application ID (ProgID and AppID). A separate Wrapper object is generated for each library defined in an IDL file. To avoid name clashes because of different Wrapper objects having the same name, we recommend that you put all programs contained in one library into one IDL file.

Example

For illustration we use the example *calc*, which can be found in subdirectory *Examples\DCOM Wrapper\calc* of the EntireX installation. In the following we refer to this directory simply as *calc*. The IDL file is named *calc.idl*. This IDL file defines a program *calc* in the library *Example*. The default Wrapper automation object name is therefore EOL.Example.

The DCOM Wrapper Wizard first generates a batch file *calc.bat* in directory *calc*. The appropriate subdirectories and GUID files are created as necessary. If you run the batch file *calc.bat*, a platform-dependent subdirectory *Example* is created in directory *calc* as described below. For each library defined in the IDL file, a subdirectory with the same name as the library is created in the platform-dependent subdirectory.

The generated Wrapper object is located in directory *calc\win32\Example* and is named *Example.dll*. If the Generate DCOM Proxy box has been checked, the directory will also contain the proxy object *pExample.dll*.

Note:
Declaring long library names within the Software AG IDL file will also result in long folder names, which in certain circumstances may cause problems.

Standard Wrapper Properties

In addition to the parameter descriptions of the remote procedures as specified in the IDL file, every automation object supports EntireX Broker, EntireX RPC and Natural RPC related properties. Methods are available to set and/or retrieve these properties. For more information, see *API Data Descriptions for the C Wrapper*.

Property Name	Description
BrokerSecurity	Get/Set Broker Kernel Security. Corresponds to Broker ACI field KERNELSECURITY. See `KERNELSECURITY`.
Codepage	Get/Set the Codepage. For sending locale strings to the broker, see *Using Internationalization with the DCOM Wrapper*.
Compression	Used to set and/or retrieve the compression value.

Property Name	Description
CompressLevel	Compression level. Valid values: N/Y/0-9. Only the first character of the string will be used for the compression. If you type YES, the character Y will be used and ES will be cut off. See also *Data Compression in EntireX Broker*.
EncryptionLevel	Encryption level used by EntireX Security.
ErrorClass	Used to retrieve the error class of last object call.
ErrorMessage	Used to retrieve the error message of last object call.
ErrorNumber	Used to retrieve the error number of last object call.
ForceLogon	Determines whether explicit or auto-logon is used by the caller.
get_StatusOfMessage(messageID)	Gets the status of the message identified by the message ID (valid for reliable RPC). See *Reliable RPC for DCOM Wrapper*.
GetVersion	Get the version of DCOM Wrapper generated object: ***n*** Total number of subscribers. **0** for Template version **1** for Library version **2** for RPC runtime version
Library	Used to set and/or retrieve the Natural library. The default for Library is the library name used in the IDL file.
MessageID	Gets the message ID (valid for reliable RPC). See *Reliable RPC for DCOM Wrapper*.
NaturalLogon	Used to set and/or retrieve the Natural logon value. The default is no Natural logon. Valid parameters are Y or N. The parameter must be a single character string. If the property has been set to Y, the next call to the Natural server will cause a Natural Logon. The properties RpcPassword and RpcUserID will be used to check the user's authentication.
NewPassword	Used only for changing the user password. The user password cannot be retrieved.
PassWord	Used only for setting the user password. The user password cannot be retrieved. The default is to use no password.
Reliable	Used to set and/or retrieve the mode for reliable RPC. See *Reliable RPC for DCOM Wrapper*.
ReliableCommit	Commit a transaction (unit of work) for reliable RPC.
ReliableRollback	Roll back a transaction (unit of work) for reliable RPC.

Property Name	Description
RpcPassword	Used only for setting the RPC user RPC password. The RPC user RPC password cannot be retrieved. The default is the value of property PassWord.
RpcUserID	Used to specify RPC user ID. It can also be used to retrieve the current RPC user ID. The default for RPC user ID is the value of property userID.
SecurityToken	Security token generated by EntireX Security and EntireX Broker after successful security validation. Do not overwrite or change it. Corresponds to the Broker ACI field `SECURITY-TOKEN`. See `SECURITY-TOKEN` and also *Overview of EntireX Security*.
ServerAddress	Used to set and retrieve the server address. The default is the value entered in the DCOM Wrapper Options dialog.
SetInfo	This method accepts named parameters. Clients can set all or any of the above attributes using this method. See *Examples* for more information.
SSLString	Used for setting the SSL parameters.
TimeOut	User can set and/or retrieve the timeout value. The default value is 50 seconds.
Token	Used to set and/or retrieve the Token.
UserID	Used to specify user ID. It can also be used to retrieve the current User ID. The default for user ID is USER.

Handling Complex Data Types

ActiveX automation technology supports only a restricted set of 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 the mapping of arrays to ActiveX data types.

Note:
If the SAFEARRAY data type is used, the system expects the array to have the correct size. It is not recommended that you leave any member of an array undefined.

This section covers the following topics:

- A Complex Structure Example
- [INOUT] Parameters
- [OUT] Parameters
- Notes on Visual Basic

A Complex Structure Example

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) = "INOUT" & (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 IARRAY    (A80/1:10)      OUT
  end-define
```

Visual Basic

```
...
dim arr()
'redim arr(10)
...
WrapperObject.BPROG 123, arr
For each strval in arr
    Document.write strval
Next
```

See *Notes on Visual Basic* below.

C/C++

In the C/C++ language, the INOUT and OUT parameters behave in the same way. See *C/C++* example for `INOUT` parameters above.

Notes on Visual Basic

- After the call that creates this array, you can check the array bounds with the Visual Basic functions LBound and UBound.
- Visual Basic arrays start with index 0. VBScript does not support the "dim myarray(... to ...)" notation. Because array sizes are checked, you must dimension your array *n* -1 when it contains *n* elements.

- IN, OUT arrays must be defined like OUT arrays and then redimensioned to the required size.
- EntireX DCOM Wrapper version 5.1 and above creates objects that support VARIANT references. Scripting languages such as VBScript pass output parameters by VARIANT references and not by 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.
- VBScript supports VARIANT reference.
- If you are using PERL for Win32 or JScript, refer to the appropriate documentation for information whether the used version supports VARIANT reference.

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
```

Standard Wrapper Methods

Standard Logon/Logoff Methods

For an explicit logon, each Wrapper object that is generated supports the methods `Logon` and `Logoff`. During logon, user ID and password are validated by EntireX Broker and EntireX Security. Logoff frees allocated resources in EntireX Broker and EntireX Security, which results in fewer bottlenecks.

Tips

- Issue a logon once for every EntireX Broker you deal with, normally at the start of the application. Do not issue separate logons for every Wrapper object when using multiple objects in an application.
- Issue a logoff whenever an EntireX Broker is not needed for a longer period, and always issue a logoff at the end of the application.
- Issue a logon and a logoff for every token used.

Visual Basic Example using Logon/Logoff

```
Dim OLEObj as Object ’ Create the Object
OLEObj.UserID = "<User ID>" ’Set the User ID
OLEObj.Password = "<Password>" ’ Set the Password
OLEObj.Logon ’ Sign on
...
OLEObj.Logoff ’ Sign off
```

Using Wrapper Objects Conversationally

This section contains the following topics:

- Programming Model
- Wrapper Methods
- Visual Basic Example using Conversations
- Tips

Programming Model

The basic method of communication for both the EntireX and the Natural RPC is non-conversational (also known as connectionless communication). Using this method, each RPC message is isolated and has no relationship to any other RPC message.

The DCOM Wrapper also supports conversational communication (also known as connection-oriented communication), where the two partners (client and server) retain a communication link over several remote procedure calls.

Conversational communication facilitates a more object-oriented design approach. In addition, a context can be maintained on the server side when a Natural RPC Server is in use. See the `DEFINE DATA CONTEXT` statement in the appropriate Natural Documentation.

Wrapper Methods

The conversation is handled by the following standard Wrapper methods:

Method Name	Description
OpenConversation	Used to open a conversation to the named server. Until one of the `CloseConversation` methods is used, all subsequent RPC messages belong to the opened conversation. A Wrapper object is either in non-conversational or conversational mode. A mixture of the two modes is not possible.
CloseConversation	Used to close the previously opened conversation. When the partner is a Natural RPC Server, it implicitly executes a database BACKOUT TRANSACTION before closing the conversation. When the partner is an EntireX RPC server, no database backout is executed. All other database operations are the responsibility of the user.
CloseConversationCommit	Used to close the previously opened conversation. When the partner is a Natural RPC Server, it implicitly executes a database END TRANSACTION before closing the conversation. When the partner is an EntireX RPC server, no database commit is executed. All other database operations are the responsibility of the user.

Visual Basic Example using Conversations

```
On Error Goto ErrHandling
Dim OLEObj as Object ’ Create Object
Set OLEObj = CreateObject("ObjectName")
OLEObj.OpenConversation ’ Open the Conversation
OLEObj.<RPC Message 1> ’ first RPC Message
OLEObj.<RPC Message 2> ’ second RPC Message
..
OLEObj.<RPC Message n> ’ n th RPC Message
OLEObj.CloseConversationCommit ’ Close the Conversation with END TRANSACTION
ErrHandling:
OLEObj.CloseConversation ’ Close the Conversation with BACKOUT TRANSACTION
```

The `OpenConversation` call establishes the conversation with the previously specified server. Assuming the RPC messages contain database update operations, the `CloseConversationCommit` makes the database modifications active by implicitly executing the END TRANSACTION operation. When an error occurs within the conversation, the database operations are backed out implicitly by the `CloseConversation` call. See also the Conversation Example in the directory *Examples\DCOM Wrapper*.

Tips

- When an END TRANSACTION or BACKOUT is needed within the conversation (without closing the conversation) simply define `Backout` and `Commit` in the IDL file as programs and implement them on the server. Backout or Commit can then be invoked as OLEObj.Backout and OLEObj.Commit.

- If you need to have a second conversation in parallel, simply define a second object in your application.

```
Dim OLEObj1 as Object ’ Create first object

Dim OLEObj2 as Object ’ Create second object
```

- Try to keep the duration of the conversations to a minimum when the Natural RPC Server is in use. Remember the server is blocked and in exclusive use by the calling client and cannot be used in parallel by other clients. Prestarting enough server replicas could improve this performance problem; using the EntireX Attach Manager would solve it. However, the EntireX RPC Server can be set up to create a replica for each new conversation. Always remember to code a `CloseConversation` call.

Method SetInfo

The method can be used to set more than one property at a time. Each property will be passed with named variables. For an example see *Setting up User ID and Timeout Value using Method* `SetInfo`.

Setting Features

To fine tune the behavior of the DCOM Wrapper object, the following features are available. If you prefer the defaults of EntireX versions up to 5.3, use `put_Feature("EXX53", true)`.

Feature	Default	Description	Note on Versions up to 5.3
StringTrimming	ON	Trim trailing space characters from the string for the received string. The String (An,Kn,AV,KV,AVn,KVn) will not be trimmed during transmission. Only the received string will be trimmed	Default: OFF. Strings not trimmed on output.
StringCutting	OFF	While using An or Kn the string will cut at the given fixed size. If this feature is switched off a string longer than the given fixed size will cause an exception. This feature does not touch the data types AVn and KVn. If an AVn or KVn was longer than the given maximum size an exception will be thrown.	Default: ON. Strings that are too long are cut.
ExactValue	ON	Check N,P,NU,PU data types after converting from a double to the data type used for internal transmission, if the value was still exactly the same. If the feature is on, it will cause an exception if the value was not exactly the same. The feature is only valid if the "Numeric/Packed decimal to double" (see *Setting DCOM Wrapper Properties*) switch was on while the DCOM object was being generated. See also *IDL Data Types*.	Default: OFF. No verification of any modification during value conversion (string/number to number).

Syntax

```
put_Feature("<feature>", <true|false>)
get_Feature("<feature>") return a Boolean value
```

where <feature> is one of the features in the table above.

VB6 Example

```
Dim STrim as Boolean
obj.put_Feature "StringTrimming", True
STrim = obj.get_Feature("StringTrimming")
```

Shared C Runtime Environment

During the generation process, the DCOM Wrapper object links the shared C Runtime library (CRT) of the used C/C++ compiler environment. The version and the name of the C Runtime library depends on the vendor and version of the C/C++ compiler used. The required shared C Runtime library must be distributed together with the DCOM Wrapper object.

The requirement of the runtime environment for your C/C++ compiler is described in the product documentation of your C/C++ compiler.

C Runtime Component of Microsoft Visual C++

The shared C Runtime (CRT) DLL was distributed by Microsoft in the past as a shared system component. The C Runtime libraries of newer Microsoft Visual C++ compilers are no longer considered system files; therefore, the C Runtime libraries have to be distributed with any application that relies on them.

C++ Compiler	C Runtime
Microsoft Visual C++ .NET 2002	Msvcr70.dll
Microsoft Visual C++ .NET 2003	Msvcr71.dll
Microsoft Visual C++ .NET 2005	Msvcr80.dll
Microsoft Visual C++ .NET 2008	Msvcr90.dll
Microsoft Visual C++ .NET 2010	Msvcr100.dll

For additional information, please see Microsoft Knowledge Base Article - 326922.

Registering a Wrapper Object

Each ActiveX automation server must be registered (in the Windows registry) so that it is recognized as an ActiveX object. When you run the Wizard, the generated Wrapper object is registered automatically. You can also register an automation object manually. This is necessary if you want to transfer the Wrapper object to another machine, or when you move the generated object to another directory.

To register the Wrapper object

1. Run the program Regsvr32.exe with the full name of the Wrapper object (generated DLL) as parameter. Regsvr32.exe is part of Microsoft Visual C++. A copy is installed in the Windows system directory.

2. Make sure the path includes the directory *<drive>:\Program Files\Common Files\Software AG*. Otherwise Regsvr32 returns an error such as:
   `LoadLibrary ("xyz.dll") failed. GetLastError returns 0x0000007e..`

To unregister the Wrapper object

- Run `Regsvr32` with the `/u` switch. This removes all registry entries for the Wrapper object. You should do this before deleting the generated object, otherwise the system registry will contain unwanted entries.

Deployment of Wrapper Objects

To use a Wrapper object on a machine other than the machine on which you generated the object

1. Make sure that EntireX Runtime is installed on the machine on which you want to use the Wrapper object.
2. Copy the generated object and then register the Wrapper object as described under *Registering a Wrapper Object* above.

Using Wrapper Objects with DCOM

You can use objects generated by the DCOM Wrapper with DCOM on Windows platforms. A Wrapper object installed on one Windows machine that performs Remote Procedure Calls via EntireX Broker can be accessed from other clients via DCOM.

To enable use of DCOM

- Check the **Generate DCOM proxy** box in the *EntireX Workbench* **File > Properties > DCOM Wrapper** before generating the object.

The generated object (e.g. *Example.dll*) must be installed on the machine that will access the Broker (see *Deployment of Wrapper Objects*). This object is ready to be used as a DCOM server. The security/identity settings for this object can be changed with the *DCOMCNFG.EXE* utility.

The proxy object (e.g. *pExample.dll*) must be registered on every DCOM client machine. The purpose of the proxy object is twofold:

- To register and unregister the Wrapper object on the client machines (registration on a DCOM client machine is different from registration on a DCOM server machine).
- To provide the type library for applications running on the client side.

For more information, see *Proxy Objects with the DCOM Wrapper*.