

webMethods EntireX

EntireX DCOM Wrapper

Version 9.6

April 2014

This document applies to webMethods EntireX Version 9.6.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Copyright © 1997-2014 Software AG, Darmstadt, Germany and/or Software AG USA, Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors..

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA, Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://documentation.softwareag.com/legal/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third-Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Document ID: EXX-EEXDCOMWRAPPER-96-20140628

Table of Contents

EntireX DCOM Wrapper	vii
1 Introduction to the DCOM Wrapper	1
Scope	2
Prerequisites for Using DCOM Wrapper Objects	2
2 Using the DCOM Wrapper	3
Starting the DCOM Wrapper	4
Setting DCOM Wrapper Preferences	4
Setting DCOM Wrapper Properties	6
Generating a DCOM Client	8
Generating a DCOM Proxy	8
3 Using the DCOM Wrapper in Command-line Mode	9
Command-line Options	10
Examples for Generating DCOM	11
Further Examples	12
4 Generated DCOM Wrapper Objects	13
Supported Data Types	14
Code Generation Process	14
Location of DCOM Wrapper Objects	15
Standard Wrapper Properties	15
Handling Complex Data Types	17
Calling Remote Procedures as Functions	22
Standard Wrapper Methods	23
Shared C Runtime Environment	27
Registering a Wrapper Object	27
Deployment of Wrapper Objects	28
Using Wrapper Objects with DCOM	28
5 Using DCOM Wrapper Objects	31
Properties	32
Set_<Property>	32
Get_<Property>	33
Properties and Groups	35
Using Generated Object Methods	43
Handling Arrays	44
Handling Groups	48
Handling Periodic Groups	51
6 Proxy Objects with the DCOM Wrapper	55
Deploying Proxy Objects	56
Using Wrapper Objects with DCOM	56
Enabling Use of DCOM	56
Accessing and Registering the Wrapper Proxy	57
Specifying the Location of the DCOM Server Object	57
DCOMCFG (DCOM Utility)	57
7 Using the DCOM Wrapper with LotusScript	61

Creating and Destroying Objects	62
Working with Arrays	62
Working with Group(s)	64
Working with Arrays of Groups	65
8 Using DCOM Wrapper Objects with Web Scripting Languages	67
Hints	68
Examples	69
9 Software AG IDL to .NET Mapping	71
Mapping IDL Data Types to .NET Data Types	72
Mapping Library Name and Alias	74
Mapping Program Name and Alias	75
Mapping Parameter Names	75
Mapping Fixed and Unbounded Arrays	76
Mapping Groups and Periodic Groups	76
Mapping Structures	76
Mapping the Direction Attributes IN, OUT and INOUT	77
Mapping the ALIGNED Attribute	77
Calling Servers as Procedures or Functions	77
10 Reliable RPC for DCOM Wrapper	79
Introduction to Reliable RPC	80
Writing a Client	81
Writing a Server	83
Broker Configuration	83
11 Tips and Tricks for the DCOM Wrapper	85
IDL Parameter Definitions	86
Groups in IDL Parameter Definition	86
Arrays in Groups	86
ActiveX Application Calling a DCOM Wrapper Method	86
Problem with the #import Clause of the Client Application	87
Syntax Errors	87
ActiveX Automation Server	88
Restrictions on Parameter Names	88
Language-dependent Restrictions on the Client Side	88
Using DCOM Wrapper Object with Microsoft Visual Studio .NET	88
Using N, NU, P, PU Data Types	90
12 Using Alias Names with the DCOM Wrapper	91
Alias for Library Names	92
Alias for Program Names	93
13 Visual Basic Example	95
14 Data Type Binary with the DCOM Wrapper	99
Mapping Data Type Binary	100
Index Order	100
Example	101
15 DCOM Wrapper Examples of Various Data Types	105
Data Type A<nn>	107

Data Type A<nn>/<nn>	107
Data Type AV	108
Data Type AV/V,V	109
Data Type AV/<nn>	110
Data Type AV/<nn>,<nn>	111
Data Type AV/<nn>,<nn>,<nn>	112
Data Type AV/V,V,V	113
Data Type I<nn>/<nn>	114
Data Type I2/V,V	115
Data Type I<nn>/<nn>,<nn>	116
Data Type I<nn>/V,V,V	117
Data Type NU<nn>/V,V	118
Data Type P<nn>.<nn>/V,V	119
Data Type L/V	120
Data Type D	121
Data Type D/V,V	122
Data Type F4/V	123
Data Type F8/V,V	124
Data Type B1/<nn>	125
Data Type B<nn>/<nn>	126
Data Type BV	127
Data Type BV/<nn>	128
Data Type B<nn>	129
Data Type BV/V	130
Data Type BV<nn>	131
Data Type B<nn>/V	132
Data Type BV/V,V	133
Data Type BV/<nn>,<nn>,<nn>	134
Data Type BV/V,V,V	135
Data Type Group	136
Data Type Group/<nn>	137
Data Type Group/<nn> with Nested Group /<nn>	138
Data Type Group/<nn>,<nn>	139
Data Type Array Nested in Group	139
Data Type Struct	140
Data Type Struct/<nn>	141
Data Type Struct/V	142
Data Type Struct/V,V	144
Data Type Struct/V,V,V	145
16 Using Arrays of Variable Sizes with the DCOM Wrapper	147
Description	148
Methods for Arrays of Variable Sizes	148
Group Definition for Arrays of Variable Sizes	150
17 Using Structures with the DCOM Wrapper	151
Overview	152

- Software AG IDL File 152
- Using Structures 156
- 18 Writing Applications with the DCOM Wrapper 159
 - Tracing 160
 - Using Internationalization with the DCOM Wrapper 160
 - Setting Transport Methods for the DCOM Wrapper 161
- 19 Microsoft Visual Studio Wizard for EntireX DCOM Wrapper 163
 - Creating a new DCOM Wrapper Project 164

EntireX DCOM Wrapper

The EntireX DCOM Wrapper generates DCOM-enabled components using RPC technology. This so-called “wrapping” makes it possible to treat existing applications as ActiveX components.

<i>Introduction</i>	Introduction to the DCOM Wrapper; prerequisites.
<i>Using</i>	Describes how to start the DCOM Wrapper, set wrapper properties, generate a DCOM client, generate a DCOM proxy.
<i>Visual Studio Wizard</i>	Describes how to use the Microsoft Visual Studio Wizard for EntireX DCOM Wrapper.
<i>Command-line Mode</i>	Using the DCOM Wrapper in command-line mode.
<i>Generated Objects</i>	Describes supported data types, the code generation process, location of DCOM Wrapper objects, standard Wrapper properties, handling complex data types, calling remote procedures as functions, standard wrapper methods, registering and deploying wrapper objects, using wrapper objects with DCOM.
<i>Using Wrapper Objects</i>	Describes properties, properties and groups, how to use generated object methods, how to handle arrays, groups and periodic groups.
<i>Proxy Objects</i>	Describes the deployment of wrapper objects, how to use wrapper objects with DCOM, how to enable the use of DCOM, how to access and register the wrapper proxy, how to specify the location of DCOM server objects (DCOM Utility).
<i>LotusScript</i>	Describes how to use LotusScript to create and destroy objects, and working with arrays and groups.
<i>Web Scripting Languages</i>	Provides hints on using VBScript as scripting language and an example.
<i>Mapping IDL to .NET</i>	Describes mapping Software AG IDL data types to .NET data types.
<i>Reliable RPC</i>	Introduction to reliable RPC; writing a client and a server for Reliable RPC; Broker configuration.
<i>Tips and Tricks</i>	Tips and tricks for IDL parameter definitions, groups in an IDL parameter definition, arrays in groups, ActiveX Application calling a DCOM Wrapper method, a problem with the #import clause of the client application, syntax errors, the ActiveX Automation Server, restrictions on parameter names, language-dependent restrictions on the client side, using DCOM Wrapper object with Microsoft Visual Studio .NET.
<i>Using Alias Names</i>	Describes how to use alias names for library and program names.
<i>Visual Basic Example</i>	Provides examples on how to create a Visual Basic object with the DCOM Wrapper and how to use the properties of this object.
<i>Data Type Binary;</i>	Describes how to map data type binary.
<i>Examples of Various Data Types</i>	Provides Visual Basic examples of how programs generated with the IDL generator are used with various data types.

Using Arrays of Variable Sizes

Describes an array without fixed size and with strings of variable length, the methods and the group definition for arrays of variable sizes.

Using Structures

Describes how to use structures with the DCOM Wrapper.

Writing Applications

Describes how to enable internationalization and tracing.

1 Introduction to the DCOM Wrapper

- Scope 2
- Prerequisites for Using DCOM Wrapper Objects 2

Scope

The EntireX DCOM Wrapper generates DCOM-enabled components using RPC technology. This so-called “wrapping” makes it possible to treat existing applications as ActiveX components.

The DCOM Wrapper provides access to server applications on server platforms such as z/OS, UNIX or Windows from DCOM-enabled applications, for example ActiveX applications running on various Windows platforms.

The DCOM Wrapper uses a Software AG IDL file (Interface Definition Language) that describes the RPC interface and generates a Wrapper object from the interface description. The Wrapper object is an ActiveX automation server that exposes the remote server's procedures as methods. Internally, the object maps these methods to RPC function calls. The generated automation objects can be used from any application that is an ActiveX automation controller.

Prerequisites for Using DCOM Wrapper Objects

The 32-bit EntireX Mini Runtime must be installed on the machine where the generated DCOM Wrapper object will be executed. The version of the Mini Runtime must be the same as (or greater than) the version of EntireX that was used to generate the DCOM Wrapper object.

The installation of EntireX will not update the version of the installed EntireX Mini Runtime. See *Mini Runtime* under *Post-installation Steps under Windows* for details.



Note: If the call to the DCOM Wrapper object fails with the DCOM error “Application or object oriented error 287”, check the following:

- the used DCOM object is registered
- the 32-bit Mini Runtime is installed
- the EntireX version that was used to generate the DCOM Wrapper object is less than or equal to the version of the installed 32-bit Mini Runtime.

2 Using the DCOM Wrapper

▪ Starting the DCOM Wrapper	4
▪ Setting DCOM Wrapper Preferences	4
▪ Setting DCOM Wrapper Properties	6
▪ Generating a DCOM Client	8
▪ Generating a DCOM Proxy	8

See also [Prerequisites for Using DCOM Wrapper Objects](#).

Starting the DCOM Wrapper

To use the DCOM Wrapper functions, open the *EntireX Workbench*.

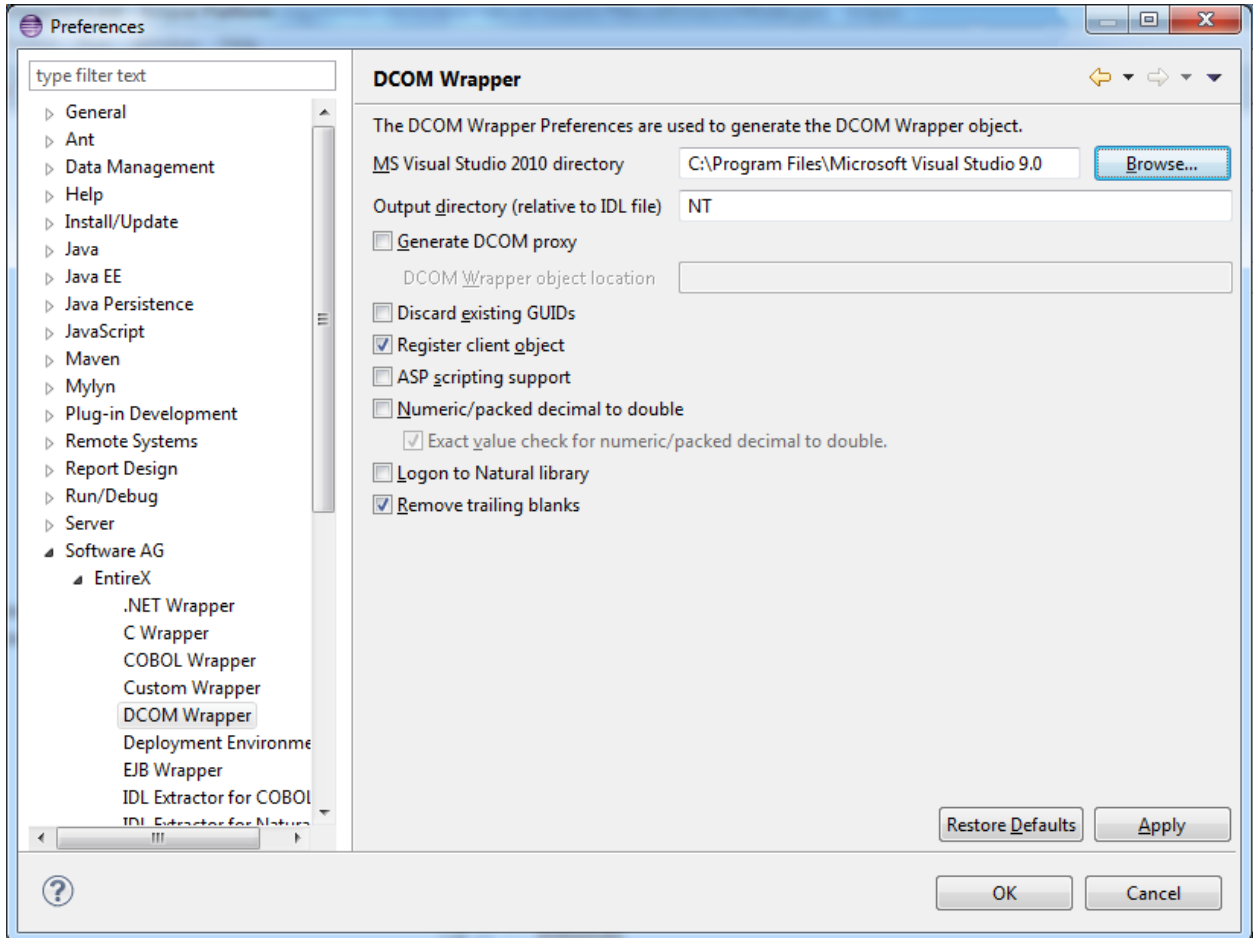
See also [Microsoft Visual Studio Wizard for EntireX DCOM Wrapper](#).

Setting DCOM Wrapper Preferences

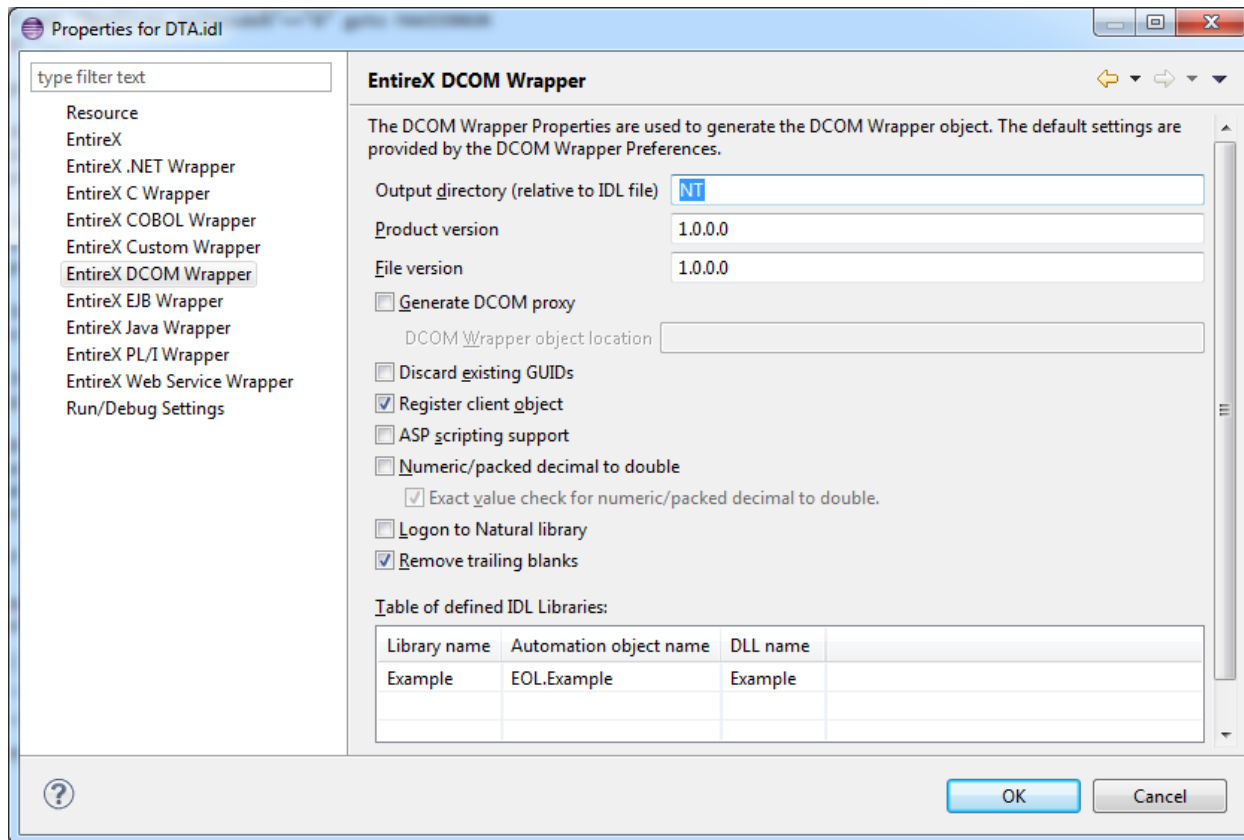
▶ To set the DCOM Wrapper Preferences

- 1 Open the DCOM Wrapper Preferences (using: **Window > Preferences > Software AG > EntireX > DCOM Wrapper**).
- 2 Ensure that the compiler path is set correctly.

For an explanation of other settings, see [Setting DCOM Wrapper Properties](#).



Setting DCOM Wrapper Properties



▶ To set the DCOM Wrapper Properties

- 1 In the tree display, select the IDL file to be processed.
- 2 Use the context menu or choose **File > Properties** to call the Properties window.

Property	Description
Output directory	<p>Specify the output directory, relative to the selected IDL file. The directory will be created if it does not exist. NT is the default to be compatible with previous versions.</p> <p>The location of the generated batch and other files depends on the name of the output directory.</p> <p><u>NT</u> Default. Batch files are generated in the same directory as the IDL file; other files are generated to subdirectory <i>NT</i>. This was the behavior in the initial version of EntireX 9.0 or earlier.</p> <p><i>other</i> Batch and other files are generated in a user-defined subdirectory.</p>

Property	Description
Product version	Specify the product version. Use the following format: <i>n.n.n.n</i>
File version	Specify the file version. Use the following format: <i>n.n.n.n</i>
Numeric/Packed decimal to double	Select this box to map Natural data types Numeric and Packed Decimal (not supported by C or Visual Basic) to data type <i>double</i> , which is supported by C and Visual Basic. If you do not select the check box, the data type is mapped as a string (BSTR). Note: Conversion of data types may lead to loss of precision. Natural supports a precision of 29 digits, while the data type double supports a precision of 15 digits. See table of Supported Data Types .
Logon to Natural Library	For Natural RPC Servers only. Check this box to advise the Natural RPC Server to logon to the Natural library for RPC requests. For a conversation, the logon to the Natural library is performed when the conversation is opened; during a conversation the Natural library cannot be changed. This is available from Natural version 2.3 and above.
Generate DCOM proxy	Check the Generate DCOM Proxy box to enable the use of DCOM.
DCOM Wrapper object location	Specify where the generated DCOM Wrapper object (for example <i>example.dll</i>) is to be located (IP-address or location name). This field is evaluated only if proxy objects (for example <i>pexample.dll</i>) are in use, otherwise it is ignored.
Discard existing GUIDs	Check this box if you have to re-generate a project: <ul style="list-style-type: none"> ■ if you want to discard all existing GUIDs ■ if you change the number of groups/structures.
Register client object	Check this box to register the client object and the wrapper after generating the object.
ASP Scripting Support	Create extended interface for ASP scripting support.
Features	See also Setting Features .
Automation Object Name	Specify the name of the registered object. The default is <i>EOL.<Library></i> . See Using Alias Names with the DCOM Wrapper .
Library Name	Specify the library of the registered object.
DLL Name	Specify the name of generated DLL file.

Generating a DCOM Client

▶ **To generate a DCOM Client**

- 1 Select the Software AG IDL file.
- 2 From the context menu, choose **Generate DCOM from Software AG IDL file**.

Generating a DCOM Proxy

▶ **To generate a DCOM Proxy**

- 1 In the *Properties*, check the box **Generate DCOM proxy**.
- 2 Select the Software AG IDL file.
- 3 From the context menu, choose **Generate DCOM from Software AG IDL file**.

3 Using the DCOM Wrapper in Command-line Mode

- Command-line Options 10
- Examples for Generating DCOM 11
- Further Examples 12

Command-line Options

See *Using the EntireX Workbench in Command-line Mode* for the general command-line syntax (note that option `-data` is not required). The table below shows the command-line options for the DCOM Wrapper.

Task	Command	Option	Description
Generate DCOM Wrapper objects for the specified IDL file(s).	-dcom:generate	-help	Display this usage message.
		-broker	The EntireX Broker.
		-server	The EntireX Service.
		-productversion	Specify the product version. Format: <number>.<number>.<number>.<number>
		-fileversion	Specify the file version. Format: <number>.<number>.<number>.<number>
		-convnpdouble	Natural data types Numeric and Packed Decimal to data type double (otherwise BSTR).
		-logonnaturallibrary	Logon to Natural library.
		-generateproxy	Generate a DCOM Proxy to enable the use of DCOM.
		-proxyserverlocation	Specify the IP address or location name of the generated DCOM Wrapper object.
		-discardexistingguids	Discard all existing GUIDs.
		-registerobject	Register the generated DOM Wrapper object after generating.
		-enableASPScripting	Create extended interface for ASP scripting support.
		-stringtrimming	Trim trailing space characters from string for the received string.
		-exactvalue	Check if data types N,P,NU or PU after converting contain the original value.
		-aonames	Specify the name of the registered objects. Format: library=aoname[,library=aoname]*
-dllnames	Specify the name of DLL files. Format: library=dllname[,library=dllname]*		
-compiler	Location of Microsoft Visual Studio 2010.		

Task	Command	Option	Description
		-output	<p>Directory for generated batch and other files (relative to IDL file). The location of the generated files depends on the name of the output directory.</p> <p><u>NT</u> Default. Batch files are generated in the same directory as IDL file; other files are generated to subdirectory <i>NT</i>. This was the behavior in the initial version of EntireX 9.0 or earlier.</p> <p><i>other</i> Batch and other files are generated in a user-defined subdirectory.</p>

Examples for Generating DCOM

```
<workbench> -dcom:generate /Demo/Example.idl
```

where *<workbench>* is a placeholder for the actual Workbench starter as described under *Using the EntireX Workbench in Command-line Mode*.

The name of the IDL file includes the project name. In the example, the project *Demo* is used. If the IDL file name describes a file within the Eclipse workspace, the name is case-sensitive.

If the first part of the IDL file name is not a project name in the current workspace, the IDL file name is used as a relative (based on the IDL file) or absolute file name in the file system. Thus, the IDL files do not need to be part of an Eclipse project.

The generated DCOM source files will be stored in parallel to the Software AG IDL file, in the generated subfolders *win32\<Library Name>*, e.g. *Demo\win32\EXAMPLE*.

Status and processing messages are written to standard output (stdout), which is normally set to the executing shell window.

Further Examples

Example 1

```
<workbench> -dcom:generate C:\Temp\example.idl
```

Uses the IDL file *C:\Temp\example.idl* and generates the DCOM source files (several .bat files, the subfolders *win32\EXAMPLE* within the different files) in parallel to the IDL file. Slashes and backslashes are permitted in the file name. Output to standard output:

```
Using workspace file:/C:/myWorkspace/.
LIBRARY = EXAMPLE
    Program = CALC
    Program = POWER
    Program = HELLO
(C) Copyright Software AG 2000-2008. All rights reserved.
Set environment for MS Visual Studio C++
...
Exit value: 0
```

Example 2

```
<workbench> -dcom:generate -help
```

or

```
<workbench> -help -dcom:generate
```

Both show a short help for the DCOM Wrapper.

4

Generated DCOM Wrapper Objects

▪ Supported Data Types	14
▪ Code Generation Process	14
▪ Location of DCOM Wrapper Objects	15
▪ Standard Wrapper Properties	15
▪ Handling Complex Data Types	17
▪ Calling Remote Procedures as Functions	22
▪ Standard Wrapper Methods	23
▪ Shared C Runtime Environment	27
▪ Registering a Wrapper Object	27
▪ Deployment of Wrapper Objects	28
▪ Using Wrapper Objects with DCOM	28

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* in the EntireX Release Notes)

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* in the IDL Editor documentation 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.
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 <i>Data Compression in EntireX Broker</i> in the general administration documentation.
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.

Property Name	Description
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.
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 <i>Overview of EntireX Security</i> in the EntireX Security documentation.
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 <code>CloseConversation</code> 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 <code>BACKOUT TRANSACTION</code> 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 <code>END TRANSACTION</code> 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 <i>IDL Data Types</i> under <i>Software AG IDL File</i> in the IDL Editor documentation.	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*](#).

5 Using DCOM Wrapper Objects

▪ Properties	32
▪ Set_<Property>	32
▪ Get_<Property>	33
▪ Properties and Groups	35
▪ Using Generated Object Methods	43
▪ Handling Arrays	44
▪ Handling Groups	48
▪ Handling Periodic Groups	51

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)"

```



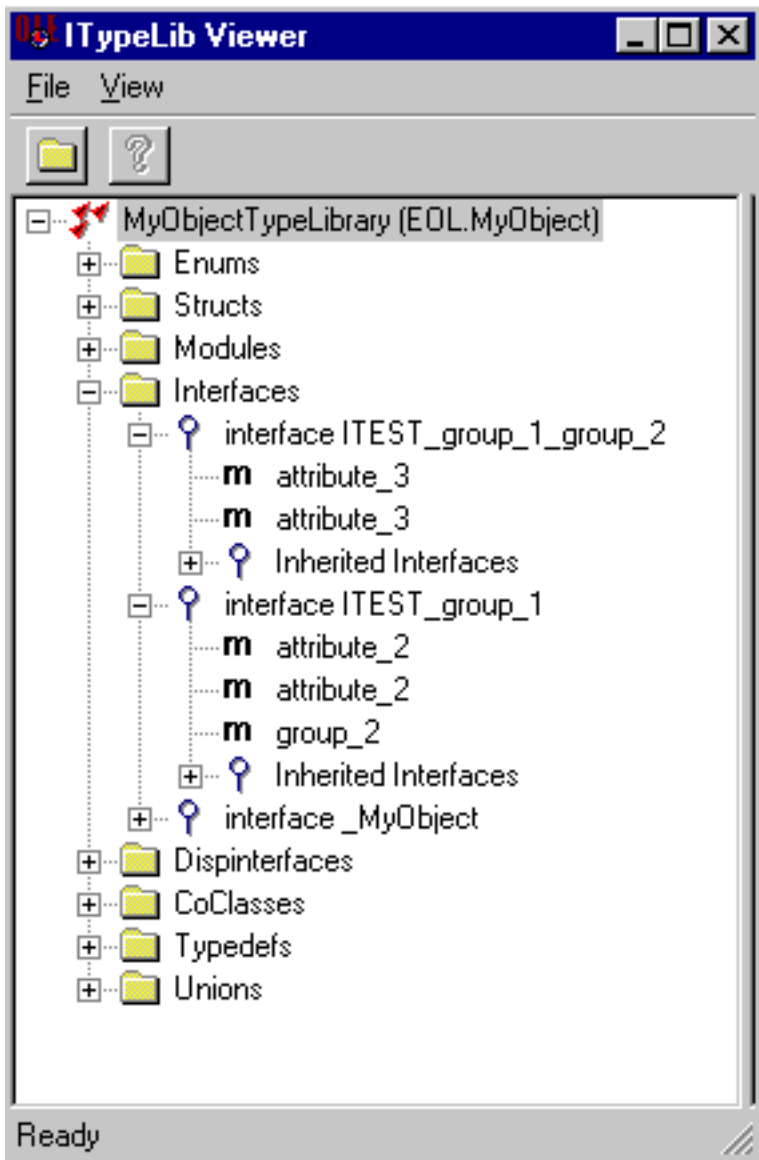
Note: Every index starts with 0.

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,
                                    &result,
                                    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 result;

    res = pDspTest_group_2->Invoke( id[0],
                                    IID_NULL,
                                    0,
                                    DISPATCH_PROPERTYGET,
                                    &params,
                                    &result,
                                    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 */
Idispach *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 i0, i1, i2 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,
                          &result,
                          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 result;
    res = pDspEXAMPLE1_mystruct ->Invoke( id[0],
                                          IID_NULL,
                                          0,
                                          DISPATCH_PROPERTYGET,
                                          &params,
                                          &result,
                                          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,
                      &result,
                      &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
```

6 Proxy Objects with the DCOM Wrapper

- Deploying Proxy Objects 56
- Using Wrapper Objects with DCOM 56
- Enabling Use of DCOM 56
- Accessing and Registering the Wrapper Proxy 57
- Specifying the Location of the DCOM Server Object 57
- DCOMCNFG (DCOM Utility) 57

Deploying Proxy Objects

▶ **To use a Wrapper object on a machine other than the machine where you generated the object**

- 1 Make sure EntireX runtime is installed on the machine where you want to use the Wrapper object.
- 2 Copy the generated object and then register the Wrapper object as described in [Registering a Wrapper Object](#).

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 Windows clients via DCOM.

Enabling Use of DCOM

Check the Generate DCOM proxy box in Advanced Options before generating the object.

The generated object (e.g. *Example.dll*) must be installed on the machine that will access the Broker (see Development of Wrapper Objects). This object is ready to use 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.

Accessing and Registering the Wrapper Proxy

▶ To access the wrapper proxy

- Copy the generated proxy object to every client machine.

Or:

Put the generated proxy object on a fileserver.

▶ To register the wrapper proxy

- Run the program Regsvr32.exe with the name of the generated proxy object as parameter. *Regsvr32.exe* is part of Microsoft Visual C++. A copy can also be found in the Windows system directory.



Note: Registration of the proxy object on the client machine does not require EntireX or any other Software AG middleware product.

Specifying the Location of the DCOM Server Object

After registering the proxy object on the DCOM client machine, specify the name of the machine where the DCOM server object is registered using the steps below:

1. With the *DCOMCNFG.EXE* utility, select the properties of the application (e.g. EOL.Example).
2. Select the Locations tab and enter the machine name under **Run application on the following computer**.

DCOMCNFG (DCOM Utility)

If access permission problems occur when running remote DCOM Wrapper objects from Windows, check the configuration using DCOMCNFG.

Server PC

■ **Default Properties**

Please check that the following default properties settings exist for all objects:

- Enable DCOM on this Computer is set.
- Authentication Level is Connect and the Impersonation Level is Identify.
- Provide Additional Security is not set.

■ **Default Security**

Default security settings should be applied globally for all objects or explicitly for the individual object. See Properties - Security. Please make sure that:

- Access Permissions contains the name Everyone or Administrators with AllowAccess.
- Launch Permissions contains the name Everyone or Administrators with AllowLaunch.
- Configuration Permissions use the default settings or your own modifications.

■ **Properties**

Please make sure that:

- General contains the Application name (for example EOL.EMPL). The application type is DLL surrogate.
- If the Default Security settings for all objects are applied as described, use the default permissions:
 - Use default access permissions.
 - Use default launch permissions.
 - Use default configuration permissions.
- If the default security settings and the settings for the individual object are different, use the following settings to access the DCOM Wrapper objects remotely:
 - Use custom access with Everyone or Administrators with AllowAccess.
 - Use custom launch with Everyone or Administrators with AllowLaunch.
 - Use custom configuration with default settings or your own modifications.
- Identity is set to interactive user.

Client

■ **Default Properties**

Please make sure that:

- Enable DCOM on this Computer is set.
- Authentication Level is Connect and the Impersonation Level is Identify.
- Provide Additional Security is not set.

■ **Default Security**

The default security settings for all objects must contain the following settings because restrictions for remote DCOM Wrapper objects are not possible:

- Access Permissions must contain the name Everyone or Administrators with AllowAccess.
- Launch Permissions must contain the name Everyone or Administrators with AllowLaunch.
- Configuration Permissions with default settings or your own modifications.

■ **Properties**

Please make sure that:

- General contains the Application name (for example EOL.EMPL).
- Application type is Remote Server.
- Remote Computer contains the host name of the server PC.

■ **Location**

Please make sure that Run Application on the Following Computer is set. Enter `< pcname >` or `< IP-address >` here.

7 Using the DCOM Wrapper with LotusScript

▪ Creating and Destroying Objects	62
▪ Working with Arrays	62
▪ Working with Group(s)	64
▪ Working with Arrays of Groups	65

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.

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
```


8

Using DCOM Wrapper Objects with Web Scripting Languages

- Hints 68
- Examples 69

Objects generated by the DCOM Wrapper can be used on Windows platforms in Web scripting languages that support ActiveX automation servers. They can be used both for client-side scripting (for example in Microsoft's Internet Explorer) or for server-side scripting (for example in Microsoft's Active Server Pages). Wrapper objects work with Microsoft's scripting engines for VBScript and JScript.

Hints

Use VBScript as Scripting Language

The DCOM Wrapper creates objects that support VARIANT references. Scripting languages such as VBScript pass output parameters by VARIANT references, not as the exactly defined type. For example, when a method of a COM interface has an out parameter of type string, VBScript passes a reference to a VARIANT to get the out parameter. DCOM Wrapper objects try to convert these references to the required reference type. VBScript supports VARIANT references.

Using DCOM Wrapper with VBScript and in ASP

VBScript and ASP allow only data type *variant*. Therefore you cannot define the `Function_result` for these application languages as an array. Only scalar data types can be used as `Function_result`, because otherwise you would not be able to access the data from VBScript and ASP. From VBScript and ASP applications you can access OUT parameters defined as an array in the IDL only if these parameters are defined correctly as array data types in the VBScript and ASP.

Sample IDL File

```
PROGRAM 'AR3_AV' is
  DEFINE DATA PARAMETER
    1 iparm          (AV/2,2,2) IN
    1 ioparm         (AV/2,2,2) IN OUT
    1 oparm          (AV/2,2,2) OUT      <- must be defined as array in VBScript
    1 Function_Result (AV) OUT          <- permitted
  END-DEFINE
```

VBScript

```
Dim oparm()      <- define as Array
```

Examples

For scripting languages where automation type references can be passed for out parameters, such as VBScript, usage is no longer restricted to functional methods (only input parameters and one function result output parameter).

IDL File

```
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
```

Example of Active Server Pages

For Active Server Pages, this example would look as follows:

```
<% set oo=server.createobject("eol.example")%>
<%= oo.userid %>
<br>
<%= oo.calc(' - ',200,8) %>
<br>
```

Known Problem and Solution

If VBScript is used in ASP to transfer arrays between the client and the server application, memory leaks can occur. To avoid this, do not use an array definition in level 1 of the IDL file. Instead, move your array definition inside a group and use the index access to array members.

Example:

```
library 'ASPLib' is
program 'AspProg' is
define data parameter
1 myArray    (A105/10)
end-define
```

change to:

```
library 'ASPLib' is
program 'AspProg' is
define data parameter
1 myEnvelope
2 myArray    (A105/10)
end-define

set/get item in array (Visual Basic)
' set
' <object name>.<access to group>.<array name>_indexAccess(<index list>) = <value>
  <object name>.myEnvelope.myArray_indexAccess( index ) = <value>

' get
' <variable> = <object name>.<access to group>.<array name>_indexAccess(<index list>)
var = <object name>.myEnvelope.myArray_indexAccess( index )
```

9 Software AG IDL to .NET Mapping

- Mapping IDL Data Types to .NET Data Types 72
- Mapping Library Name and Alias 74
- Mapping Program Name and Alias 75
- Mapping Parameter Names 75
- Mapping Fixed and Unbounded Arrays 76
- Mapping Groups and Periodic Groups 76
- Mapping Structures 76
- Mapping the Direction Attributes IN, OUT and INOUT 77
- Mapping the ALIGNED Attribute 77
- Calling Servers as Procedures or Functions 77

Mapping IDL Data Types to .NET Data Types

The table below lists the metasymbols and informal terms that are used for the Software AG IDL.

- The metasymbols [and] surround optional lexical entities.
- The informal term “number” is a sequence of numeric characters, for example 123.

Software AG IDL	Description	.NET Data Types	Note
A1	Alphanumeric	char or String/StringBuilder	1, 5
<i>Anumber</i>	Alphanumeric	String/StringBuilder	1
AV	Alphanumeric variable length	String/StringBuilder	1
AV[<i>number</i>]	Alphanumeric variable length with maximum length	String/StringBuilder	1
B1	Binary	byte or byte[]	6
<i>Bnumber</i>	Binary	byte[]	
BV	Binary variable length	byte[]	2
BV[<i>number</i>]	Binary variable length with maximum length	byte[]	
D	Date	DateTime	3, 7
F4	Floating point (small)	float	
F8	Floating point (large)	double	
I1	Integer (small)	sbyte	
I2	Integer (medium)	short	
I4	Integer (large)	int	
<i>Knumber</i>	Kanji	String/StringBuilder	1
KV	Kanji variable length	String/StringBuilder	1
KV[<i>number</i>]	Kanji variable length with maximum length	String/StringBuilder	1
L	Logical	bool	
<i>Nnumber</i> [. <i>number</i>]	Unpacked decimal	decimal	
<i>NUnumber</i> [. <i>number</i>]	Unpacked decimal unsigned	decimal	
<i>Pnumber</i> [. <i>number</i>]	Packed decimal	decimal	
<i>PUnumber</i> [. <i>number</i>]	Packed decimal unsigned	decimal	
T	Time	DateTime	4, 7



Notes:

1. `System.String` for `direction` in, otherwise `System.Text.StringBuilder` if `Default` is used for parameter `ATOSTRING`. If `String` is used for `ATOSTRING`, `System.String` is used everywhere, and if `StringBuilder` is used for `ATOSTRING`, `System.Text.StringBuilder` is used everywhere. See *Using the .NET Wrapper*.
2. Unsigned integer ranging from 0 to 255.
3. Count of days AD (anno domini, after the birth of Christ). The valid range is from 1.1.0001 up to 28.11.2737 (only the date part of `DateTime` is used).
4. Count of tenths of a second AD (Anno Domini, after the birth of Christ). The valid range is from 1.1.0001 00:00:00.0 up to 16.11.3168 09:46:39 plus 0.9 seconds.
5. If `-D A1TOCHAR=1` is defined in the `erxid1` call, `A1` is mapped to `char`, otherwise to `String/StringBuilder`.
6. If `-D B1TOBYTE=1` is defined in the `erxid1` call, `B1` is mapped to `byte`, otherwise to `byte[]`.
7. The Natural `DATE` type allows for the value 01.01.0000 to denote an undefined date. In order to avoid the .NET runtime throwing an exception when attempting to assign the invalid date value 01.01.0000 to a .NET `DateTime` variable, the .NET runtime converts an incoming neutral date/time value 01.01.0000 00:00:00.0 into the special .NET `DateTime` value `DateTime.MaxValue - 1 tick` (that is 31.12.9999:23:59:59.9999998). When this value is passed to the EntireX runtime to be sent to an EntireX RPC service, it is converted back into the neutral RPC date/time value 01.01.0000 00:00:00.0.

Please also note the hints and restrictions on the IDL data types valid for all programming language bindings as described under *IDL Data Types* under *Software AG IDL File* in the IDL Editor documentation.

Mapping Library Name and Alias

The library name as specified in the IDL file is sent from a client to the server. Special characters are not replaced. The library alias is not sent to the server.

In the RPC server, the IDL library name sent may be used to locate the target server. See *Locating and Calling the Target Server* in the platform-specific administration or RPC server documentation.

The name of the .NET server assembly must match the library name.

The library name as given in the IDL file is used to compose the names of the generated output files. See `library-definition` under *Software AG IDL Grammar* in the *IDL Editor* documentation. Therefore the allowed characters are restricted by the underlying file system. The name is composed from `<library-name>.idl` to `<library-name>.cs` as default. The name of the client stub file can be changed by using the `-F` option of the `erxidl` command. See *Using the .NET Wrapper in IDL Compiler Command-line Mode*.

In accordance with the C# conventions, the class name is built as follows with the default setting `-PSANITIZE`:

- The initial character and characters following one of the special characters '#', '\$', '&', '+', '-', '_', '.', '/' and '@' are converted to uppercase.
- All other characters are converted to lowercase.
- The special characters '#', '\$', '&', '+', '-', '_', '.', '/' and '@' are removed.

Other special characters used in the library name are not changed and may lead to problems with your underlying file system and to compile errors.

If there is an alias for the library name in the `library-definition`, this alias is used “as is” to form the class name. Therefore, this alias must be a valid C# class name. To fully control the output, use alias names and do not use `SANITIZE`.

Examples:

```
MY-CLASS to MyClass (class)
```

```
MY-CLASS alias YOUR_CLASS to YOUR_CLASS(class)
```

Mapping Program Name and Alias

The program name is sent from a client to the server. Special characters are not replaced. The program alias is not sent to the server.

In the RPC server, the IDL program name sent is used to locate the target server. See *Locating and Calling the Target Server* in the platform-specific administration or RPC server documentation.

The program names as given in the IDL file are mapped to methods within the generated C# sources. See *program-definition* under *Software AG IDL Grammar* in the *IDL Editor* documentation.

In accordance with the C# conventions method names are built as follows with the default setting `-PSANITIZE`:

- Characters are converted to lowercase with the following exceptions
 - The special characters '#', '\$', '&', '+', '-', '_', '.', '/' and '@' are removed
 - The character following one of the special characters is converted to uppercase.

Other special characters used in the program name are not changed and may lead to compile errors.

If there is an alias for the program name in the *program-definition* under *Software AG IDL Grammar* in the *IDL Editor* documentation, this alias is used “as is” for the method name. Therefore, this alias must be a valid C# method name. To fully control the output, use alias names and do not use `SANITIZE`.

Examples:

```
MY-PROGRAM to MyProgram (method).
```

```
MY-PROGRAM alias YOUR_PROGRAM to YOUR_PROGRAM(method).
```

Mapping Parameter Names

The parameter names as given in the *parameter-data-definition* of the IDL file are mapped to parameters of the generated C# methods.

In accordance with the C# conventions the parameter names are built as follows with the default setting `-PSANITIZE`:

- Characters are converted to lowercase except
 - The special characters '#', '\$', '&', '+', '-', '_', '.', '/' and '@' are removed
 - The character following one of those special characters is converted to uppercase.

IDL files that use C# keywords (e.g. `string` or `float`) as parameter names are not supported. Do not use C# keywords such as `string` or `float` as parameter names. Modify your IDL file accordingly.

To fully control the output do not use `SANITIZE`.

Example:

MY-PARAM to `myParam` (parameter)

Mapping Fixed and Unbounded Arrays

Arrays in the IDL file are mapped to C# arrays. If an array value does not have the correct number of dimensions or elements, this will result in an exception. If the value `null` (null pointer) is used as an input parameter (for `IN` and `INOUT` parameters), an array will be instantiated by the runtime.

Mapping Groups and Periodic Groups

Groups in the IDL file are mapped to C# classes.

The namespace for group classes is `SoftwareAG.EntireX.NETWrapper.Generated.filename.Groups` on the client side, and `SoftwareAG.EntireX.NETWrapper.Server.libraryname.Groups` on the server side.

Mapping Structures

Structures in the IDL file are mapped to C# classes.

The namespace for structure classes is `SoftwareAG.EntireX.NETWrapper.Generated.filename.Structs` on the client side, and `SoftwareAG.EntireX.NETWrapper.Server.libraryname.Structs` on the server side.

See [Mapping Groups and Periodic Groups](#).

Mapping the Direction Attributes IN, OUT and INOUT

- IN parameters are implemented as normal parameters of the generated C# class method.
- OUT parameters are implemented as out parameters of the generated C# class method.
- INOUT parameters are implemented as ref parameters of the generated method.

Note that only the direction information of the top-level fields (level 1) is relevant. Group fields always inherit the specification from their parent. A different specification is ignored.

See `attribute-list` under *Software AG IDL Grammar* in the *IDL Editor* documentation for the syntax on how to describe attributes within the IDL file and refer to the `direction` attribute.

Mapping the ALIGNED Attribute

Not supported.

Calling Servers as Procedures or Functions

The IDL syntax allows definitions of procedures only. It does not have the concept of a function. A function is a procedure which, in addition to the parameters, returns a value. Procedures and functions are transparent between clients and server, i.e. a client using a function can call a server implemented as a procedure and vice versa.

In C# a procedure corresponds to a method with result type `void`, a function returns a value of some type.

It is possible to treat an `OUT` parameter of a procedure as the return value of a function. The .NET Wrapper generates a method with a non-void result type 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`.

In this case no function parameter is generated for this `OUT` parameter.

See the .NET Wrapper example that comes with EntireX.

10

Reliable RPC for DCOM Wrapper

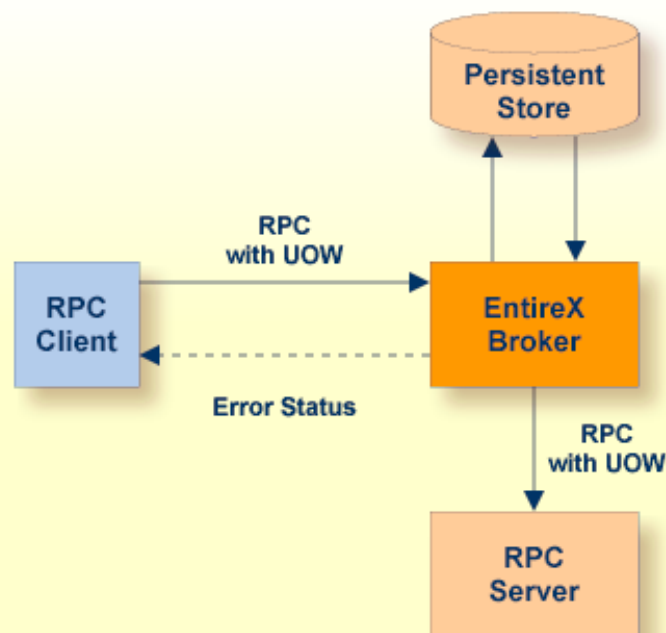
- Introduction to Reliable RPC 80
- Writing a Client 81
- Writing a Server 83
- Broker Configuration 83

Introduction to Reliable RPC

In the architecture of modern e-business applications (such as SOA), loosely coupled systems are becoming more and more important. Reliable messaging is one important technology for this type of system.

Reliable RPC is the EntireX implementation of a reliable messaging system. It combines EntireX RPC technology and persistence, which is implemented with units of work (UOWs).

- Reliable RPC allows asynchronous calls (“fire and forget”)
- Reliable RPC is supported by most EntireX wrappers
- Reliable RPC messages are stored in the Broker's persistent store until a server is available
- Reliable RPC clients are able to request the status of the messages they have sent



Reliable RPC is used to send messages to a persisted Broker service. The messages are described by an IDL program that contains only `IN` parameters. The client interface object and the server interface object are generated from this IDL file, using the EntireX DCOM Wrapper.

Reliable RPC is enabled at runtime. The client has to set one of two different modes before issuing a reliable RPC request:

- `AUTO_COMMIT`
- `CLIENT_COMMIT`

While `AUTO_COMMIT` commits each RPC message implicitly after sending it, a series of RPC messages sent in a unit of work (UOW) can be committed or rolled back explicitly using `CLIENT_COMMIT` mode.

The server is implemented and configured in the same way as for normal RPC.

Writing a Client

All methods for reliable RPC are available on the interface object. See [Standard Wrapper Properties](#) for details. The methods are

- `RPCService.Reliable` (put and get)
- `RPCService.ReliableCommit`
- `RPCService.ReliableRollback`
- `RPCService.MessageID`
- `RPCService.get_StatusOfMessage`

Create Broker object and interface object:

```
// DCOM Wrapper Object
MAILClass mail;
mail.Logon();
```

Disable reliable RPC:

```
mail.Reliable = mail.RELIABLE_OFF;
```

Enable reliable RPC with `AUTO_COMMIT` or `CLIENT_COMMIT`:

```
mail.Reliable = mail.RELIABLE_AUTO_COMMIT;
mail.Reliable = mail.RELIABLE_CLIENT_COMMIT;
```

The first RPC message:

```
mail.SENDMAIL("mail receiver", "Subject 1", "Text 1");
```

Check the status: get the message ID first and use it to retrieve the status:

```
String messageID = mail.MessageID;  
String messageStatus = mail.get_StatusOfMessage(messageID);  
System.out.println("Status: " + messageStatus + ", id: " + messageID);
```

The second RPC message:

```
mail.SENDMAIL("mail receiver", "Subject 2", "Text 2");
```

Commit the two messages:

```
mail.ReliableCommit();
```

Check the status again for the same message ID:

```
messageStatus = mail.get_StatusOfMessage(messageID);  
System.out.println("Status: " + messageStatus + ", id: " + messageID);
```

The third RPC message.

```
mail.SENDMAIL("mail receiver", "Subject 3", "Text 3");
```

Check the status: get the new message ID and use it to retrieve the status:

```
messageID = mail.MessageID();  
messageStatus = mail.get_StatusOfMessage(messageID);  
System.out.println("Status: " + messageStatus + ", id: " + messageID);
```

Roll back the third message and check status:

```
mail.ReliableRollback();  
messageStatus = mail.getStatusOfMessage(messageID);  
System.out.println("Status: " + messageStatus + ", id: " + messageID);  
mail.logoff();
```

Limitations

- All program calls that are called in the same transaction (CLIENT_COMMIT) must be in the same IDL library.
- It is not allowed to switch from CLIENT_COMMIT to AUTO_COMMIT in a transaction.
- Messages (IDL programs) must have only IN parameters.

Writing a Server

The server implementation consist of the four classes

- Abstract<IDL library name>Server
- <IDL library name>
- <IDL library name>Server
- <IDL library name>Stub

Add your implementation to the class <IDL library name>Server. There are no server-side methods for reliable RPC. The server does not send back a message to the client. The server can run deferred, thus client and server do not necessarily run at the same time. If the server fails, it throws an exception. This causes a cancel of the transaction (unit of work inside the broker) and the error code is written to the user status field of the unit of work.

Broker Configuration

A Broker configuration with `PSTORE` is recommended. This enables the Broker to store the messages for more than one Broker session. These messages are still available after Broker restart. The attributes `STORE`, `PSTORE`, and `PSTORE-TYPE` in the Broker attribute file can be used to configure this feature. The lifetime of the messages and the status information can be configured with the attributes `UWTIME` and `UWSTAT-LIFETIME`. Other attributes such as `MAX-MESSAGES-IN-UOW`, `MAX-UOWS` and `MAX-UOW-MESSAGE-LENGTH` may be used in addition to configure the units of work. See *Broker Attributes* in the platform-independent administration documentation.

The result of the function `RPCService.get_StatusOfMessage` depends on the configuration of the unit of work status lifetime in the EntireX Broker configuration. If the status is not stored longer than the message, the function returns the error code 00780305 (no matching UOW found).

11

Tips and Tricks for the DCOM Wrapper

▪ IDL Parameter Definitions	86
▪ Groups in IDL Parameter Definition	86
▪ Arrays in Groups	86
▪ ActiveX Application Calling a DCOM Wrapper Method	86
▪ Problem with the #import Clause of the Client Application	87
▪ Syntax Errors	87
▪ ActiveX Automation Server	88
▪ Restrictions on Parameter Names	88
▪ Language-dependent Restrictions on the Client Side	88
▪ Using DCOM Wrapper Object with Microsoft Visual Studio .NET	88
▪ Using N, NU, P, PU Data Types	90

This chapter provides the following tips and tricks for using EntireX DCOM Wrapper:

See also *Using DCOM Wrapper Objects with Web Scripting Languages*.

IDL Parameter Definitions

The parameter definition in the IDL file must match exactly the parameter data area of the Natural subprogram with respect to format, length, dimensions, direction, number and order of parameters.

Groups in IDL Parameter Definition

Because the number of required GUIDs may change if the parameter definition in the IDL file changes with respect to groups, you must deregister the old object and delete the *g<library>.h* file before (re-)generating the object.

Arrays in Groups

From EntireX 5.3.1.10 on, methods for better array support are available for arrays defined in groups.

```
set property
<object name>.<access to group>.<array name>_indexAccess(<index list>) = value

get property
<variable> = <object name>.<access to group>.<array name>_indexAccess(<index list>)
```

ActiveX Application Calling a DCOM Wrapper Method

When the ActiveX application calls a DCOM Wrapper method, the parameters must match exactly:

- the number of parameters
- the parameter type (string, int etc.)
- how these parameters are passed to the called method:
 - IN parameters should be passed by value;
 - OUT and INOUT parameters should be passed by reference.

If in doubt, look at the type library of the generated Wrapper object. Use the Microsoft C++ tool OLE/COM Object Viewer (*oleview.exe*) to inspect the type library, or look at the generated IDL file in the generated object directory.

Problem with the #import Clause of the Client Application

If the dll generated by EntireX DCOM Wrapper is imported by a client application, the compiler can exit with error “error C2011: 'IServiceProvider': 'struct' type redefinition”. If this happens, add the following to the #import clause:

```
rename("IServiceProvider", "IServiceProviderX")
```

Example:

```
#import "<path>\DCOMWrapperObject.dll" no_namespace ↵  
rename("IServiceProvider", "IServiceProviderX")
```

If the dll generated by EntireX DCOM Wrapper is imported by a client application and this application is an ATL application, it may be necessary to import the dll type library with the following statement:

```
#include <dispex.h>  
#import "DCOMWrapperObject.tlb" no_namespace exclude("IDispatchEx")  
exclude("IServiceProvider") rename ("IDispatch", "IDispatchEx")
```

Syntax Errors

If syntax errors occur in the IDL file, the most likely cause is the use of reserved words as parameters. Not only the keywords of IDL (library, program, etc.) are reserved, but also all valid format-length combinations. For example, you cannot use "A1" as the name of a parameter.

If syntax errors occur during compilation of the generated C++ sources, a likely cause is the use of reserved C++ words as parameter names in the IDL file, or a library that contains a program with the same name.

ActiveX Automation Server

If you generate an ActiveX automation server a second time, the existing GUIDs will be reused. To prevent this, delete the header file *g<library>.h* in the generated object directory.

Restrictions on Parameter Names

Parameter names used in the IDL file conflict with the Visual C++ MIDL compiler, so avoid keywords used in the MIDL definition. The DCOM Wrapper uses the MIDL compiler from EntireX Version 6.1.1.8 on (only Windows platforms). The changes were necessary because of the internal change from the IDispatch to the IDispatchEx interface to support scripting languages correctly.

Language-dependent Restrictions on the Client Side

On the client side, language-dependent restrictions may occur independent of the functionality offered by the DCOM Wrapper object.

Using DCOM Wrapper Object with Microsoft Visual Studio .NET

While using the Wrapper-generated objects, the .NET application throws an error message `queryinterface <interface name> failed`. To prevent this, uncheck **ASP scripting support** in the *Setting DCOM Wrapper Properties*.

If the Extended Interface was used, the client application has to make sure that the Wrapper-generated object was called from an STA thread. In a ASP.NET application, the client application can use the `aspcompat` attribute on your ASP.NET page or if the application, such as Web service, does not provide an attribute such as `aspcompat`, use an STA thread in your application.

ASP.NET Example

Include the attribute `aspcompat` into the file `WebForm.aspx`.

```
<%@ Page aspcompat=true Language="vb".....
```

Web Service Example

```
' Create new thread class
Public Class STAThreadClass
    Dim Obj As EXXDCOMTypeLibrary.ProgClass
    Dim sUser As String

    Public Sub Run()
        ' Set thread state to STA
        System.Threading.Thread.CurrentThread.ApartmentState = ←
        Threading.ApartmentState.STA
        Obj = New EXXDCOMTypeLibrary.ProgClass()

        Try
            Obj.ServerAddress = "localhost@RPC/SRV1/CALLNAT"
            Obj.TEST(sUser)
            ' catch an error, if one thrown
        Catch ex As Exception
            ReturnValues.ErrorText = "Error: " + ex.ToString
        End Try
        Obj = Nothing
    End Sub
End Class

' main web service function
<WebMethod()> Public Function DoCall()

    Dim STA As New STAThreadClass()

    ' create the thread
    Dim InstanceCaller As New Thread(New ThreadStart(AddressOf STA.Run))

    ' start the thread and wait for execution
    InstanceCaller.Start() 'start thread
    InstanceCaller.Join() 'wait for called STA thread

End Function
```

Using N, NU, P, PU Data Types

When using N, NU, P or PU data types, the only valid decimal character is the decimal point.

Examples:

1234.34	OK
1234,34	not allowed
123,400.00	not allowed

Only the decimal point "." is allowed!

12

Using Alias Names with the DCOM Wrapper

- Alias for Library Names 92
- Alias for Program Names 93

The Software AG IDL Compiler supports alias names for libraries and programs.

You can use alias names to replace cryptic names of programs, libraries, etc. on the client side with descriptive names. For example, you can define the alias name *CalculateTable* for the program name *X301G1* in example *Group*.

Alias for Library Names

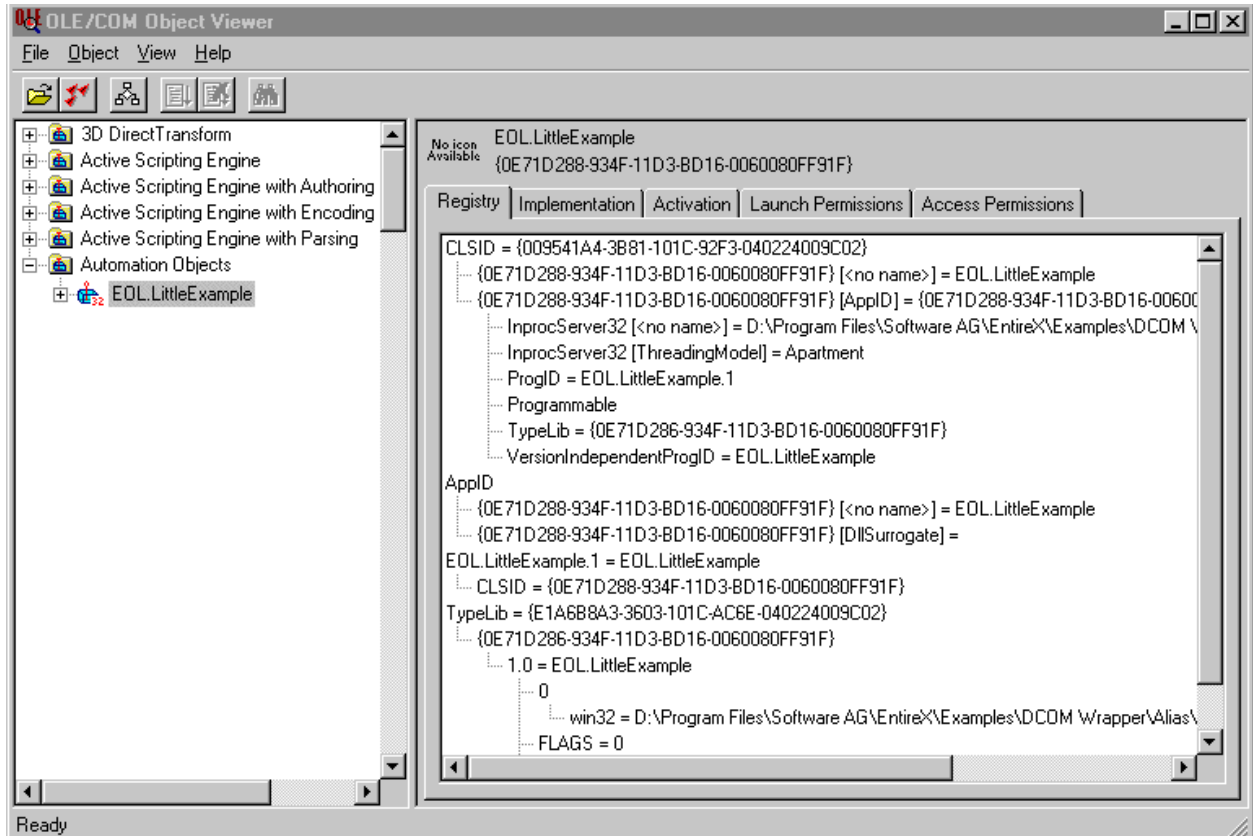
With the DCOM Wrapper you can enter alias names, see [Setting DCOM Wrapper Properties](#), and you can add them to the IDL file. If no alias is defined for the library, the default is EOL.< library name > (as displayed in the Naming Options window). If the alias in the IDL file differs from that in the saved settings for this IDL file, the wizard asks which alias you want to use.

Declaring an Alias for a Library Name

```
Library 'EXAMPLE ':'EOL.LittleExample' Is
```

The code line shows the part of the Software AG IDL file that defines an alias name for the library. For the RPC component, the name EXAMPLE is used. A client must use the name EOL.LittleExample to refer to generated automation objects.

The library alias name is used for the registry entry.



Alias for Program Names

```

Program 'CALC': 'Calculator' Is
...
Program 'SQUARE' Is
...

```

These lines of an IDL file are the head lines of two different program definitions. The server-side programs are called with `CALC` and `SQUARE` from the automation object generated by the DCOM Wrapper. The client calling this automation object uses the methods `Calculator` and `SQUARE`.

13 Visual Basic Example

This chapter provides examples on how to create a Visual Basic object with the DCOM Wrapper and how to use the properties of this object.

Define an Object :

```
Dim OLEObj as Object
```

Create an Object :

```
Set OLEObj = CreateObject("<Object Name>")
```

Retrieve User ID :

```
Dim sUserID As String  
sUserID = OLEObj.UserID
```

Retrieve RPC User ID :

```
Dim sRpcUserID As String  
sRpcUserID = OLEObj.RpcUserID
```

Retrieve Timeout Value :

```
Dim lTimeOut As Integer  
lTimeOut = OLEObj.TimeOut
```

Retrieve and display Server Address :

```
MsgBox "Server Address : " & OLEObj.ServerAddress
```

Retrieve Token :

```
Dim sToken As String  
sToken = OLEObj.Token
```

Retrieve Library :

```
Dim sLibrary As String  
sLibrary = OLEObj.Library
```

Retrieve Compression :

```
Dim sCompression As String  
sCompression = OLEObj.Compression
```

Retrieve Natural Logon :

```
Dim sNaturalLogon As String  
sNaturalLogon = OLEObj.NaturalLogon
```

Retrieve Error Class :

```
Dim sErrorClass As Long  
sErrorClass = OLEObj.ErrorClass
```

Retrieve Error Number :

```
Dim sErrorNumber As Long  
sErrorNumber = OLEObj.ErrorNumber
```

Retrieve Error Message :

```
Dim sErrorMessage As String  
sErrorMessage = OLEObj.ErrorMessage
```

Display Error Message :

Use the Description property of Visual Basic's Error object, for example:

```
On Error GoTo frmload_ErrProc  
...  
frmload_ErrProc:  
MsgBox Err.Description, Err.Source
```

Set User ID :

```
OLEObj.UserID = "<User ID>"
```

Set Password :

```
OLEObj.Password = "<Password>"
```

Set NewPassword :

```
OLEObj.NewPassword = "<NewPassword>"
```

Set RPC User ID :

```
OLEObj.RpcUserID = "<RpcUser ID>"
```

Set RPC Password :

```
OLEObj.RpcPassword = "<RpcPassword>"
```

Set Server Address :

```
OLEObj.ServerAddress = "<Server Address>"
```

Set Time Out Value :

```
OLEObj.TimeOut = <Timeout value>
```

Set Token :

```
OLEObj.Token = "<Token>"
```

Set Library :

```
OLEObj.Library = "<Library Name>"
```

Set Compression :

```
OLEObj.Compression = "<Compression Type>"  
<Compression Type> ::= '<blank>' | '2'
```


Set Natural Logon :

```
OLEObj.NaturalLogon = "<Natural Logon>"
<Natural Logon> ::= 'Y' | 'N'
```

Open Conversation :

```
OLEObj.OpenConversation
```

Close Conversation with Backout:

```
OLEObj.CloseConversation
```

Close Conversation with Commit :

```
OLEObj.CloseConversationCommit
```

Broker Logon :

```
OLEObj.Logon
```

Broker Logoff :

```
OLEObj.Logoff
```

Setting up User ID and Timeout Value using Method SetInfo:

```
OLEObj.SetInfo UserID:=tbUserID.Text, TimeOut:=30
```

Similarly, other named parameters can be used for setting connection and user-specific information. Other named parameters are as follows :

Named Parameter	Purpose
UserID	Sets User ID, maximum 32 characters long.
Password	Sets Password, maximum 32 characters long.
NewPassword	Sets NewPassword, maximum 32 characters long.
Token	Sets Token, maximum 32 characters long.
BrokerID	Sets Broker ID, maximum 32 characters long.
ServerName	Sets Server Name, maximum 32 characters long.
ServerClass	Sets Server Class, maximum 32 characters long.
ServiceName	Sets Service Name, maximum 32 characters long.
TimeOut	Sets Timeout value, timeout value is validated at server end. Cannot be less than or equal to 0.
RpcUserID	Sets RPC User ID, maximum 32 characters long.
RpcPassword	Sets RPC Password, maximum 32 characters long.
ForceLogon	Determines whether explicit or auto-logon is used by the caller.
EncryptionLevel	Encryption level used by EntireX Security.
SSLString	Used for setting SSL parameters.
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.

Named Parameter	Purpose
	See also <i>Data Compression in EntireX Broker</i> in the general administration documentation.

14 Data Type Binary with the DCOM Wrapper

▪ Mapping Data Type Binary	100
▪ Index Order	100
▪ Example	101



Note: Mapping and handling of data type binary (IDL file format B) changed in version 5.2.1.6 of the DCOM Wrapper. To avoid conversions of any binary data, the mapping of data type binary was changed from automation type BSTR to automation type unsigned char.

Mapping Data Type Binary

Version 5.3.1 and above

Software AG IDL		Visual Basic	C++	Note
B1		Byte	unsigned char	
B n	1 GB $\geq n > 1$	Byte()	SAFEARRAY(unsigned char)	1
B $n/i_1, i_2, i_3$	1 GB $\geq n \geq 1$	Byte()	SAFEARRAY(unsigned char)	1, 2
BV1		Byte	unsigned char	
BV n	1 GB $\geq n > 1$	Byte()	SAFEARRAY(unsigned char)	1
BV $n/i_1, i_2, i_3$	1 GB $\geq n \geq 1$	Byte()	SAFEARRAY(unsigned char)	1, 2
BV	Variable size ≤ 1 GB	Byte()	SAFEARRAY(unsigned char)	1
BV/ i_1, i_2, i_3	Variable size ≤ 1 GB	Byte()	SAFEARRAY(unsigned char)	1

Notes

1. 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.
2. Depending on n , the SAFEARRAY has 3 dimensions ($n = 1$) or 4 dimensions ($n > 1$).

Index Order

The n ($n > 1$) length value is used as the rightmost index, for $n = 1$ the index n is dropped.

Example:

```
Bn / i1,i2,i3    --> i1,i2,i3,n
B1 / i1,i2,i3    --> i1,i2,i3
```



Note: If the IDL file contains data of type binary, the code generated by the EntireX DCOM Wrapper 5.2.1.6 is not compatible to the code generated by older versions of the EntireX DCOM Wrapper. If the IDL file does not contain data of type binary, the codes are compatible.

Example

```
Library 'Bintest' Is
Program 'bintest1':'binary1' Is
  Define Data Parameter
    1 LittleBin (B1)
    1 MaxBin (B126)
    1 BigBin (B16/20)
    1 VeryBigBin (B10/3,4,5)
  End-Define

Program 'bintest2':'binary2' Is
  Define Data Parameter
    1 Group
    2 BigBin (B126/20)
    1 PerGroup (/2,3)
    2 AnotherBin (B30/10)
  End-Define
```

Visual Basic

```
Dim obj As Object

...

Set obj = CreateObject("Eol.BinTest")

...

' Init parameter of program "binary1"
Dim vbLittleBin As Byte
Dim vbMaxBin() As Byte
Dim vbBigBen() As Byte
Dim vbVeryBigBin() As Byte

' Parameter of program "binary2"
' --
```

```
' others
Dim checksum As Long

' redim Parameter of program "binary1"
ReDim vbMaxBin(1 To 126)
ReDim vbBigBin(1 To 20, 1 To 16)
ReDim vbVeryBigBin(1 To 3, 1 To 4, 1 To 5, 1 To 10)

'''' binary1 ''''
' Fill in binary data
vbLittleBin = &H0

For i1 = 1 To 126
    vbMaxBin(i1) = i1
Next i1

For i1 = 1 To 20
    For i2 = 1 To 16
        vbBigBin(i1, i2) = 130 + i2 + i1
    Next i2
Next i1

For i1 = 1 To 3
    For i2 = 1 To 4
        For i3 = 1 To 5
            For i4 = 1 To 10
                vbVeryBigBin(i1, i2, i3, i4) = 130 + i2 + i1
            Next i4
        Next i3
    Next i2
Next i1

obj.binary1 vbLittleBin, vbMaxBin, vbBigBin, vbVeryBigBin

' Reading of binary data
' here for example:
' takes all binary data values and add them to variable checksum
checksum = 0

For i1 = 1 To 3
    For i2 = 1 To 4
        For i3 = 1 To 5
            For i4 = 1 To 10
                checksum = checksum + vbVeryBigBin(i1, i2, i3, i4)
            Next i4
        Next i3
    Next i2
Next i1
```

```

...

''' binary2 '''
Dim barray20_126() As Byte
Dim barray10_30() As Byte

ReDim barray20_126(19, 125)
ReDim barray10_30(9, 29)

For i1 = 0 To 19
    For i2 = 0 To 125
        ' fill in some values
        barray20_126(i1, i2) = i1 + i2
    Next i2
Next i1

obj.binary2_group.bigbin = barray20_126 '(B126/20)

For i1 = 0 To 1
    For i2 = 0 To 2

        For i3 = 0 To 9
            For i4 = 0 To 29
                ' fill in some values
                barray10_30(i3, i4) = 255 - i1 - i2
            Next i4
        Next i3

        obj.binary2_perigroup(i1, i2).anotherbin = barray10_30 '(B30/10)
    Next i2
Next i1

obj.binary2 obj.binary2_group, obj.binary2_perigroup_all

' Reading of binary data
' here for example:
' takes all binary data values and add them to variable checksum

barray20_126 = obj.binary2_group.bigbin '(B126/20)

checksum = 0
For i1 = 0 To 19
    For i2 = 0 To 125
        checksum = checksum + barray20_126(i1, i2)
    Next i2
Next i1

For i1 = 0 To 1
    For i2 = 0 To 2
        barray10_30 = obj.binary2_perigroup(i1, i2).anotherbin
    
```

```
    For i3 = 0 To 9
      For i4 = 0 To 29
        checksum = checksum + barray10_30(i3, i4)
      Next i4
    Next i3

  Next i2
Next i1
```


15 DCOM Wrapper Examples of Various Data Types

▪ Data Type A<nn>	107
▪ Data Type A<nn>/<nn>	107
▪ Data Type AV	108
▪ Data Type AV/V	109
▪ Data Type AV/<nn>	110
▪ Data Type AV/<nn>,<nn>	111
▪ Data Type AV/<nn>,<nn>,<nn>	112
▪ Data Type AV/V,V,V	113
▪ Data Type I<nn>/<nn>	114
▪ Data Type I2/V,V	115
▪ Data Type I<nn>/<nn>,<nn>	116
▪ Data Type I<nn>/V,V,V	117
▪ Data Type NU<nn>/V,V	118
▪ Data Type P<nn>.<nn>/V,V	119
▪ Data Type L/V	120
▪ Data Type D	121
▪ Data Type D/V,V	122
▪ Data Type F4/V	123
▪ Data Type F8/V,V	124
▪ Data Type B1/<nn>	125
▪ Data Type B<nn>/<nn>	126
▪ Data Type BV	127
▪ Data Type BV/<nn>	128
▪ Data Type B<nn>	129
▪ Data Type BV/V	130
▪ Data Type BV<nn>	131
▪ Data Type B<nn>/V	132
▪ Data Type BV/V,V	133
▪ Data Type BV/<nn>,<nn>,<nn>	134
▪ Data Type BV/V,V,V	135
▪ Data Type Group	136
▪ Data Type Group/<nn>	137

- Data Type Group/<nn> with Nested Group /<nn> 138
- Data Type Group/<nn>, <nn> 139
- Data Type Array Nested in Group 139
- Data Type Struct 140
- Data Type Struct/<nn> 141
- Data Type Struct/V 142
- Data Type Struct/V,V 144
- Data Type Struct/V,V,V 145

This chapter provides Visual Basic examples of how programs generated with the IDL generator are used with various data types.

Data Type A<nn>

IDL

```
Program 'ConstA' Is
  Define Data Parameter
    1 Client      (A80) In
    1 Mail        (A80) In Out
  End-Define
```

Visual Basic Code

```
Function GetMail(S1 As String, S2 As String) As String
  Dim Client As String * 80
  Dim Mail As String * 80

  Client = S1
  Mail = S2

  obj.ConstA Client, Mail ' Perform call to dcom object

  GetMail = Mail
End Function
```

Data Type A<nn>/<nn>

IDL

```
Program 'ConstAN' Is
  Define Data Parameter
    1 Client      (A80/3) In
    1 Mail        (A80/2) In Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail(S1 As String, S2 As String) As String()  
    Dim Client(2) As String  
    Dim Mail(1) As String  
    Dim tempStr As String * 80 ' string with fixed size  
  
    For i = 0 To UBound(Client)  
        tempStr = S1  
        Client(i) = tempStr  
    Next  
  
    For i = 0 To UBound(Mail)  
        tempStr = S2  
        Mail(i) = tempStr  
    Next  
  
    obj.ConstAN Client, Mail ' Perform call to dcom object  
  
    GetMail = Mail  
End Function
```

Data Type AV

IDL

```
Program 'ArrayAV' Is  
    Define Data Parameter  
        1 Client (AV) In  
        1 Mail (AV) In Out  
    End-Define
```

Visual Basic Code

```
Private Function GetMail(S1 As String, S2 As String) As String()  
    Dim Client As String  
    Dim Mail As String  
  
    Client = S1  
    Mail = S2  
  
    obj.ArrayAV Client, Mail ' Perform call to dcom object  
  
    GetMail = Mail  
End Function
```

Data Type AV/V,V

IDL

```

Program 'ArrayAVVV' Is
  Define Data Parameter
    1 Client      (AV/V,V)  In
    1 Mail        (AV/V,V)  In Out
  End-Define

```

Visual Basic Code

```

Private Function GetMail(S1 As String, S2 As String) As Variant()
  Dim Client() As Variant      ' ** define array parameter
  Dim Mail() As Variant        ' ** define array parameter

  ReDim Client(4)
  ReDim Mail(2)

  Dim m_str() As String

  For i = 0 To UBound(Client)
    ReDim m_str(2)
    For j = 0 To UBound(m_str)
      m_str(j) = Str(i) & Str(j) & S1 ' assign a string
    Next j
    Client(i) = m_str ' assign a new array
  Next i

  For i = 0 To UBound(Mail)
    ReDim m_str(1)
    For j = 0 To UBound(m_str)
      m_str(j) = Str(i) & Str(j) & S2 ' assign a string
    Next j
    Mail(i) = m_str ' assign a new array
  Next i

  obj.ArrayAVVV Client, Mail      ' Perform call to dcom object

  GetMail = Mail
End Function

```

Data Type AV/<nn>

IDL

```
Program 'ArrayAVnn' Is
  Define Data Parameter
    1 Client      (AV/2)  In
    1 Mail        (AV/5)  In Out
    1 Server      (AV/7)  Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail(S1 As String, S2 As String) As String()
  Dim Client() As String
  Dim Mail() As String
  Dim Server() As String

  ReDim Client(1)      '2 elements
  ReDim Mail(4)        '5 elements

  For i = 0 To UBound(Client)
    Client(i) = S1 & " " & CStr(i)
  Next i

  For i = 0 To UBound(Mail)
    Mail(i) = S2 & " " & CStr(i)
  Next i

  obj.ArrayAVnn Client, Mail, Server ' Perform call to dcom object

  GetMail = Server
End Function
```

Data Type AV/<nn>,<nn>

IDL

```

Program 'ArrayAVnnnn' Is
  Define Data Parameter
    1 Client      (AV/2,3)  In
    1 Mail        (AV/5,7)  In Out
    1 Server      (AV/7,9)  Out
  End-Define

```

Visual Basic Code

```

Private Function GetMail(S1 As String, S2 As String) As Variant()
  Dim Client() As String
  Dim Mail()   As String
  Dim Server() As String

  ReDim Client(1, 2)      '2,3 elements
  ReDim Mail(4, 6)       '5,7 elements

  For i = 0 To UBound(Client, 1)
    For j = 0 To UBound(Client, 2)
      Client(i, j) = S1 & " " & CStr(i) & " " & CStr(j)
    Next j
  Next i

  For i = 0 To UBound(Mail, 1)
    For j = 0 To UBound(Mail, 2)
      Mail(i, j) = S2 & " " & CStr(i) & " " & CStr(j)
    Next j
  Next i

  obj.ArrayAVnnnn Client, Mail, Server ' Perform call to dcom object

  GetMail = Server
End Function

```

Data Type AV/<nn>,<nn>,<nn>

IDL

```
Program 'ArrayAVnnnnnn' Is
  Define Data Parameter
    1 Client      (AV/2,3,4)  In
    1 Mail        (AV/5,7,8)  In Out
    1 Server      (AV/9,6,3)  Out
  End-Define
```

See illustration for client (AV/2,3,4) under *Example of Arrays with Fixed Bounds* under *Software AG IDL Grammar* in the *IDL Editor* documentation.

Visual Basic Code

```
Private Function GetMail(S1 As String, S2 As String) As Variant()
  Dim Client() As String
  Dim Mail()   As String
  Dim Server() As String

  ReDim Client(1, 2, 3)      '2,3,4 elements
  ReDim Mail(4, 6, 7)       '5,7,8 elements

  For i = 0 To UBound(Client, 1)
    For j = 0 To UBound(Client, 2)
      For k = 0 To UBound(Client, 3)
        Client(i, j, k) = S1 & " " & CStr(i) & " " & CStr(j) & " "
& CStr(k)
      Next k
    Next j
  Next i

  For i = 0 To UBound(Mail, 1)
    For j = 0 To UBound(Mail, 2)
      For k = 0 To UBound(Mail, 3)
        Mail(i, j, k) = S2 & " " & CStr(i) & " " & CStr(j) & " "
& CStr(k)
      Next k
    Next j
  Next i

  obj.ArrayAVnnnnnn Client, Mail, Server ' Perform call to dcom object

  GetMail = Server
End Function
```


Data Type AV/V,V,V

IDL

```

Program 'ArrayAVVVV' Is
  Define Data Parameter
    1 Client      (AV/V,V,V)  In
    1 Mail        (AV/V,V,V)  In Out
    1 Server      (AV/V,V,V)  In Out
  End-Define

```

See illustration for client (AV/V,V,V) under *Three-dimensional Array with Variable Upper Bounds* under *Software AG IDL Grammar* in the *IDL Editor* documentation.

Visual Basic Code

```

Private Function GetMail(S1 As String, S2 As String) As Variant()
  Dim Client() As Variant
  Dim Mail()   As Variant
  Dim Server() As Variant

  ReDim Client(4)
  ReDim Mail(2)

  Dim m_str() As String
  Dim m_Array() As Variant

  For i = 0 To UBound(Client)
    ReDim m_Array(8)
    For j = 0 To UBound(m_Array)
      ReDim m_str(2)
      For k = 0 To UBound(m_str)
        m_str(k) = S1 & Str(i) & Str(j) & S1 ' assign a string
      Next k
      m_Array(j) = m_str
    Next j
    Client(i) = m_str ' assign a new array
  Next i

  For i = 0 To UBound(Mail)
    ReDim m_Array(8)
    For j = 0 To UBound(m_Array)
      ReDim m_str(2)
      For k = 0 To UBound(m_str)
        m_str(k) = S1 & Str(i) & Str(j) & S2 ' assign a string
      Next k
      m_Array(j) = m_str
    Next j
  Next i

```

```
Next j
Mail(i) = m_str ' assign a new array
Next i

obj.ArrayAVVV Client, Mail, Server ' Perform call to dcom object

GetMail = Mail
End Function
```

Data Type I<nn>/<nn>

IDL

```
Program 'ConstInn' Is
  Define Data Parameter
    1 Client (I4/4) In
    1 Mail (I4/5) In Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail(L1 As Long, L2 As Long) As Long()
  Dim Client(3) As Long ' define I4 as Long, I2 as Integer and I1 as Integer
  Dim Mail(4) As Long

  For i = 0 To UBound(Client)
    Client(i) = L1
  Next

  For i = 0 To UBound(Mail)
    Mail(i) = L2
  Next

  obj.ConstInn Client, Mail ' Perform call to dcom object

  GetMail = Mail
End Function
```

Data Type I2/V,V

IDL

```

Program 'Int2VV' Is
  Define Data Parameter
    1 Client    (I2/V,V) In
    1 Mail      (I2/V,V) In Out
  End-Define

```

Visual Basic Code

```

Private Function GetMail(I1 As Integer, I2 As Integer) As Integer()
  Dim Client() As Variant    ' ** define array parameter
  Dim Mail() As Variant      ' ** define array parameter

  Dim ClientValue() As Integer
  Dim MailValue() As Integer

  ReDim Client(8)
  ReDim Mail(3)
  ReDim ClientValue(4)
  ReDim MailValue(5)

  For i = 0 To UBound(Client)
    For j = 0 To UBound(ClientValue)
      ClientValue(j) = i * j + I1 ' write any data into the elements
    Next j
    Client(i) = ClientValue      ' assign the new array to an element of the ←
main array
  Next i

  For i = 0 To UBound(Mail)
    For j = 0 To UBound(MailValue)
      MailValue(j) = i * j + I2 ' write any data into the elements
    Next j
    Mail(i) = MailValue         ' assign the new array to an element of the ←
main array
  Next i

  obj.Int2VV Client, Mail      ' Perform call to dcom object

  GetMail = Mail
End Function

```

Data Type I<nn>/<nn>,<nn>

IDL

```
Program 'Int4NN' Is
  Define Data Parameter
    1 Client      (I4/5,4) In
    1 Mail        (I4/6,3) In Out
    1 Server      (I4/3,2) Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail(I1 As Long, I2 As Long) As Long()
  Dim Client() As Long      ' define I4 as Long, I2 as Integer and I1 as Integer
  Dim Mail() As Long
  Dim Server() As Long

  ReDim Client(4,3)
  ReDim Mail(5,2)

  For i = 0 To UBound(Client, 1)
    For j = 0 To UBound(Client, 2)
      Client(i, j) = i * j + I1 ' write any data into the elements
    Next j
  Next i

  For i = 0 To UBound(Mail, 1)
    For j = 0 To UBound(Mail, 2)
      Mail(i, j) = i * j + I2 ' write any data into the elements
    Next j
  Next i

  obj.Int4VV Client, Mail, Server ' Perform call to dcom object

  GetMail = Server
End Function
```

Data Type I<nn>V,V,V

IDL

```

Program 'Int2VVV' Is
  Define Data Parameter
    1 Client      (I2/V,V,V)  In
    1 Mail        (I2/V,V,V)  In Out
    1 Server      (I2/V,V,V)  Out
  End-Define

```

Visual Basic Code

```

Private Function GetMail(I1 As Integer, I2 As Integer) As Integer()
  Dim Client() As Variant
  Dim Mail() As Variant
  Dim Server() As Variant

  Dim ClientWrap() As Variant
  Dim MailWrap() As Variant

  Dim ClientValue() As Integer ' define I4 as Long, I2 as Integer and I1 as Integer
  Dim MailValue() As Integer

  ReDim Client(4)
  ReDim Mail(5)

  ReDim ClientWrap(10)
  ReDim MailWrap(23)

  ReDim ClientValue(200)
  ReDim MailValue(238)

  For i = 0 To UBound(Client)
    For j = 0 To UBound(ClientWrap)
      For k = 0 To UBound(ClientValue)
        ClientValue(k) = k + i * j + I1 ' write any data into the elements
      Next k
      ClientWrap(j) = ClientValue
    Next j
    Client(i) = ClientWrap
  Next i

  For i = 0 To UBound(Mail)
    For j = 0 To UBound(MailWrap)
      For k = 0 To UBound(MailValue)
        MailValue(k) = k + i * j + I2 ' write any data into the elements
      Next k
    Next j
  Next i

```

```

    MailWrap(j) = MailValue
    Next j
    Mail(i) = MailWrap
    Next i

    obj.Int2VVV Client, Mail, Server ' Perform call to dcom object

    GetMail = Server
End Function

```

Data Type NU<nn>/V,V

IDL

```

Program 'NUnVV' Is
  Define Data Parameter
    1 Client      (NU21/V,V)  In
    1 Mail        (NU21/V,V)  In Out
  End-Define

```

Visual Basic Code

```

Private Function GetMail() As String()
  Dim Client() As Variant ' ** define array parameter
  Dim Mail() As Variant ' ** define array parameter

  Dim ClientValue() As String
  Dim MailValue() As String

  ReDim Client(8)
  ReDim Mail(3)
  ReDim ClientValue(4)
  ReDim MailValue(5)

  For i = 0 To UBound(Client)
    For j = 0 To UBound(ClientValue)
      ClientValue(j) = "55456698" ' write any data into the elements
    Next j
    Client(i) = ClientValue ' assign the new array to an element of the ←
main array
  Next i

  For i = 0 To UBound(Mail)
    For j = 0 To UBound(MailValue)
      MailValue(j) = "548799875465" ' write any data into the elements
    Next j
    Mail(i) = MailValue ' assign the new array to an element of the ←

```

```

main array
  Next i

  obj.NUnVV Client, Mail      ' Perform call to dcom object

  GetMail = Mail
End Function

```

Data Type P<nn>.<nn>/V,V

IDL

```

Program 'PnXVV' Is
  Define Data Parameter
    1 Client      (P12.2/V,V) In
    1 Mail        (P12.2/V,V) In Out
  End-Define

```

Visual Basic Code

```

Private Function GetMail() As String()
  Dim Client() As Variant      ' ** define array parameter
  Dim Mail()   As Variant      ' ** define array parameter

  Dim ClientValue() As String
  Dim MailValue() As String

  ReDim Client(8)
  ReDim Mail(3)
  ReDim ClientValue(4)
  ReDim MailValue(5)

  For i = 0 To UBound(Client)
    For j = 0 To UBound(ClientValue)
      ClientValue(j) = "1234568" & "." & "65" ' write any data into the
elements
    Next j
    Client(i) = ClientValue      ' assign the new array to an element of the ←
main array
  Next i

  For i = 0 To UBound(Mail)
    For j = 0 To UBound(MailValue)
      MailValue(j) = "8799875" & "." & "32" ' write any data into the
elements
    Next j
    Mail(i) = MailValue        ' assign the new array to an element of the ←

```

```
main array
  Next i

  obj.PnXVV Client, Mail      ' Perform call to dcom object

  GetMail = Mail
End Function
```

Data Type L/V

IDL

```
Program 'LV' Is
  Define Data Parameter
    1 Client      (L/V)  In
    1 Mail        (L/V)  In Out
    1 Server      (L/V)  Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Boolean()
  Dim Client() As Boolean
  Dim Mail() As Boolean
  Dim Server() As Boolean

  ReDim Client(8)
  ReDim Mail(3)

  For i = 0 To UBound(Client)
    Client(i) = True
  Next i

  For i = 0 To UBound(Mail)
    Mail(i) = False
  Next i

  obj.LV Client, Mail, Server      ' Perform call to dcom object

  GetMail = Mail
End Function
```


Data Type D

IDL

```
Program 'DateProc' Is
  Define Data Parameter
    1 Client    (D) In
    1 Mail      (D) In Out
    1 Server    (D) Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As String
  Dim Client As Date
  Dim Mail As Date
  Dim Server As Date

  Dim s As String

  ' set the input data with the current date
  Client = Date
  Mail = Date

  obj.DateProc Client, Mail, Server

  ' work with the returned data
  s = Mail & " " & Server

  GetMail = s
End Function
```

Data Type D/V,V

IDL

```
Program 'DateProcV' Is
  Define Data Parameter
    1 Client    (D/V,V)  In
    1 Mail      (D/V,V)  In Out
    1 Server    (D/V,V)  Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As String
  Dim Client() As Variant
  Dim Mail() As Variant
  Dim Server() As Variant

  Dim Value() As Double ' Use a Double instead of a Date datatype
  Dim s As String

  Redim Client(5)
  Redim Value(12)
  For i = 0 To UBound(Client)
    For j = 0 To UBound(Value)
      ' set the input data with the current date
      Value(j) = Date
    Next j
    Client(i) = Value
  Next i

  Redim Mail(5)
  Redim Value(12)
  For i = 0 To UBound(Mail)
    For j = 0 To UBound(Value)
      ' set the input data with the current date
      Value(j) = Date
    Next j
    Mail(i) = Value
  Next i

  obj.DateProcV Client, Mail, Server

  ' work with the returned data
  s = ""
  For i = 0 To UBound(Mail)
    Value = Mail(i)
    For j = 0 To UBound(Value)
```

```

        s = s & CDate(Value(j)) ' Convert the data to a Date data type if ↵
necessary
        Next j
        s = s & " "
    Next i

    s = ""
    For i = 0 To UBound(Server)
        Value = Server(i)
        For j = 0 To UBound(Value)
            s = s & CDate(Value(j))
        Next j
        s = s & " "
    Next i

    GetMail = s
End Function

```

Data Type F4/V

IDL

```

Program 'ProcF4V' Is
    Define Data Parameter
        1 Client      (F4/V)  In
        1 Mail        (F4/V)  In Out
        1 Server      (F4/V)  Out
    End-Define

```

Visual Basic Code

```

Private Function GetMail() As String
    Dim Client() As Single
    Dim Mail()   As Single
    Dim Server() As Single

    Dim s As String

    Redim Client(5)
    For i = 0 To UBound(Client)
        Client(i) = 1.5542 + (i * 1.57)
    Next i

    Redim Mail(28)
    For i = 0 To UBound(Mail)
        Mail(i) = -1.554 + (i * 1.57)
    Next i

```

```
obj.ProcF4V Client, Mail, Server

s = ""
For i = LBound(Server) To UBound(Server)
    s = " " & Str(Server(i))
Next i

GetMail = s
End Function
```

Data Type F8/V,V

IDL

```
Program 'ProcF8VV' Is
  Define Data Parameter
    1 Client      (F8/V,V)  In
    1 Mail        (F8/V,V)  In Out
    1 Server      (F8/V,V)  Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As String
  Dim Client() As Variant
  Dim Mail()   As Variant
  Dim Server() As Variant

  Dim Value() As Double
  Dim s As String

  Redim Client(5)
  Redim Value(33)
  For i = 0 To UBound(Client)
    For j = 0 To UBound(Value)
      Value(j) = 1.5587 + (j * 1.57) + i
    Next j
    Client(i) = Value
  Next i

  Redim Mail(42)
  Redim Value(3)
  For i = 0 To UBound(Mail)
    For j = 0 To UBound(Value)
      Value(j) = -5021.327 + (j * 3.889) - i
    Next j
```

```

    Mail(i) = Value
Next i

obj.ProcF8VV Client, Mail, Server

s = ""
For i = LBound(Server) To UBound(Server)
    Value = Server(i)
    For j = LBound(Value) To UBound(Value)
        s = " " & Str(Value(j))
    Next j
Next i

GetMail = s
End Function

```

Data Type B1/<nn>

IDL

```

Program 'Bin1nn' Is
    Define Data Parameter
        1 Client      (B1/100) In
        1 Mail        (B1/100) In Out
    End-Define

```

Visual Basic Code

```

Private Function GetMail(B1 As Byte, B2 As Byte) As Byte()
    Dim Client() As Byte
    Dim Mail() As Byte

    ReDim Client(99)
    ReDim Mail(99)

    ' Client
    For j = 0 To UBound(Client)
        Client(j) = j + B1
    Next j

    'Mail
    For j = 0 To UBound(Mail)
        Mail(j) = j + B2
    Next j

    obj.Bin1nn Client, Mail

```

```
GetMail = Mail  
End Function
```

Data Type B<nn>/<nn>

IDL

```
Program 'BinXnn' Is  
  Define Data Parameter  
    1 Client      (B10/12) In  
    1 Mail        (B10/12) In Out  
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Byte()  
  Dim Client() As Byte  
  Dim Mail() As Byte  
  
  ReDim Client(11, 9)  
  ReDim Mail(11, 9)  
  
  ' Client  
  For j = 0 To UBound(Client, 1)  
    For i = 0 To UBound(Client, 2)  
      Client(j, i) = j + i  
    Next i  
  Next j  
  
  'Mail  
  For j = 0 To UBound(Mail, 1)  
    For i = 0 To UBound(Mail, 2)  
      Mail(j, i) = j + i + 6  
    Next i  
  Next j  
  
  obj.BinXnn Client, Mail  
  
  GetMail = Mail  
End Function
```

Data Type BV

IDL

```
Program 'BinV' Is
  Define Data Parameter
    1 Client    (BV)  In
    1 Mail      (BV)  In Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Byte()
  Dim Client      As Variant
  Dim Mail        As Variant
  Dim ClientValue() As Byte
  Dim MailValue()  As Byte

  ReDim ClientValue(8)
  ReDim MailValue(4)

  For j = 0 To UBound(ClientValue)
    ClientValue(j) = j + 6
  Next j

  For j = 0 To UBound(MailValue)
    MailValue(j) = j + 8
  Next j

  Client = ClientValue
  Mail = MailValue

  obj.BinV Client, Mail

  GetMail = Mail
End Function
```

Data Type BV/<nn>

IDL

```
Program 'BinVnn' Is
  Define Data Parameter
    1 Client      (BV/3)  In
    1 Mail        (BV/5)  In Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Byte()
  Dim Client()      As Variant
  Dim Mail()        As Variant
  Dim ClientValue() As Byte
  Dim MailValue()   As Byte

  ReDim Client(2)   ' /3
  ReDim Mail(4)     ' /4
  ReDim ClientValue(3)
  ReDim MailValue(5)

  For i = 0 To UBound(Client)
    For j = 0 To UBound(ClientValue)
      ClientValue(j) = j + 2
    Next j
    Client(i) = ClientValue
  Next i

  For i = 0 To UBound(Mail)
    For j = 0 To UBound(MailValue)
      MailValue(j) = j + 3
    Next j
    Mail(i) = MailValue
  Next i

  obj.BinVnn Client, Mail

  GetMail = Mail
End Function
```


Data Type B<nn>

IDL

```
Program 'Binnn' Is
  Define Data Parameter
    1 Client      (B12) In
    1 Mail        (B30) In Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Variant()
  Dim Client() As Variant
  Dim Mail() As Variant

  Dim ClientValue() As Byte
  Dim MailValue() As Byte

  ReDim ClientValue(11) ' must be 11 ( = 12 elements -> B12)
  For j = 0 To UBound(ClientValue)
    ClientValue(j) = j + 6
  Next j
  Client = ClientValue

  ReDim MailValue(29) ' must be 29 ( = 30 elements -> B30)
  For j = 0 To UBound(MailValue)
    MailValue(j) = j + 6
  Next j
  Mail = MailValue

  obj.Binnn Client, Mail

  GetMail = Mail
End Function
```

Data Type BV/V

IDL

```
Program 'BinVV' Is
  Define Data Parameter
    1 Client      (BV/V)  In
    1 Mail        (BV/V)  In Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Variant()
  Dim Client() As Variant
  Dim Mail() As Variant

  Dim ClientValue() As Byte
  Dim MailValue() As Byte

  ReDim Client(5)
  ReDim ClientValue(32)
  For i = 0 To UBound(Client)
    For j = 0 To UBound(ClientValue)
      ClientValue(j) = j + i + 2
    Next j
    Client(i) = ClientValue
  Next i

  ReDim Mail(33)
  ReDim MailValue(20)
  For i = 0 To UBound(Mail)
    For j = 0 To UBound(MailValue)
      MailValue(j) = j + i + 2
    Next j
    Mail(i) = MailValue
  Next i

  obj.BinVV Client, Mail

  GetMail = Mail
End Function
```

Data Type BV<nn>

IDL

```
Program 'BinVxx' Is
  Define Data Parameter
    1 Client    (BV10)  In
    1 Mail      (BV12)  In Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Variant
  Dim Client As Variant
  Dim Mail As Variant

  Dim ClientValue() As Byte
  Dim MailValue() As Byte

  ReDim ClientValue(9) ' 9 ( = 10 elements) or less
  For j = 0 To UBound(ClientValue)
    ClientValue(j) = j + 6
  Next j
  Client(i) = ClientValue

  ReDim MailValue(11) ' 11 ( = 12 elements) or less
  For j = 0 To UBound(MailValue)
    MailValue(j) = j + 7
  Next j
  Mail(i) = MailValue

  obj.BinVxx Client, Mail

  GetMail = Mail
End Function
```

Data Type B<nn>V

IDL

```
Program 'BinnV' Is
  Define Data Parameter
    1 Client      (B4/V)  In
    1 Mail        (B2/V)  In Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Variant
  Dim Client()      As Variant
  Dim Mail()        As Variant
  Dim ClientValue() As Byte
  Dim MailValue()   As Byte

  ReDim Client(22)
  ReDim ClientValue(3) ' B4
  For i = 0 To UBound(Client)
    For j = 0 To UBound(ClientValue)
      ClientValue(j) = j + i + 2
    Next j
    Client(i) = ClientValue
  Next i

  ReDim Mail(15)
  ReDim MailValue(1) ' B2
  For i = 0 To UBound(Mail)
    For j = 0 To UBound(MailValue)
      MailValue(j) = j + i + 2
    Next j
    Mail(i) = MailValue
  Next i

  obj.BinnV Client, Mail

  GetMail = Mail
End Function
```

Data Type BV/V,V

IDL

```

Program 'BinVVV' Is
  Define Data Parameter
    1 Client      (BV/V,V)  In
    1 Mail        (BV/V,V)  In Out
    1 Server      (BV/V,V)  Out
  End-Define

```

Visual Basic Code

```

Private Function GetMail() As Variant()
  Dim Client() As Variant
  Dim Mail()   As Variant
  Dim Server() As Variant

  Dim ClientWrap() As Variant
  Dim MailWrap()   As Variant

  Dim ClientValue() As Byte
  Dim MailValue()   As Byte

  ' Define the Bounds of the Arrays
  ReDim Client(8)
  ReDim ClientWrap(12)
  ReDim ClientValue(4)

  ReDim Mail(3)
  ReDim MailWrap(6)
  ReDim MailValue(5)

  ' Setup the Arrays
  For i = 0 To UBound(Client)
    For j = 0 To UBound(ClientWrap)
      For k = 0 To UBound(ClientValue)
        ClientValue(k) = k+j+i+1 ' write any data into the elements
      Next k
      ClientWrap(j) = ClientValue
    Next j
    Client(i) = ClientWrap      ' assign the new array to an element of the main array
  Next i

  For i = 0 To UBound(Mail)
    For j = 0 To UBound(MailWrap)
      For k = 0 To UBound(MailValue)

```

```
        MailValue(k) = k+j+i+2 ' write any data into the elements
    Next k
    MailWrap(j) = MailValue
Next j
Mail(i) = MailWrap          ' assign the new array to an element of the main array
Next i

obj.BinVVV Client, Mail, Server

GetMail = Mail
End Function
```

Data Type BV/<nn>,<nn>,<nn>

IDL

```
Program 'BinVnnnnnn' Is
  Define Data Parameter
    1 Client      (BV/3,4,6)    In
    1 Mail        (BV/5,2,5)    In Out
    1 Server      (BV/2,8,2)    Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Variant()
  Dim Client() As Variant
  Dim Mail() As Variant
  Dim Server() As Variant

  Dim ClientValue() As Byte
  Dim MailValue() As Byte

  ' Define the Bounds of the Arrays
  ReDim Client(2,3,5)
  ReDim Mail(4,1,4)

  ' Setup the Arrays
  For i = 0 To UBound(Client, 1)
    For j = 0 To UBound(Client, 2)
      For k = 0 To UBound(Client, 3)
        ReDim ClientValue(9 + i + j + k)
        For m = 0 To UBound(ClientValue)
          ClientValue(m) = i + j + k + m
        Next m
        Client(i, j, k) = ClientValue ' write any data into the elements
      Next k
    Next j
  Next i
```

```

    Next j
Next i

For i = 0 To UBound(Mail, 1)
    For j = 0 To UBound(Mail, 2)
        For k = 0 To UBound(Mail, 3)
            ReDim MailValue(3 + i + j + k)
            For m = 0 To UBound(MailValue)
                MailValue(m) = i + j + k + m
            Next m
            Mail(i, j, k) = MailValue ' write any data into the elements
        Next k
    Next j
Next i

obj.BinVnnnnnn Client, Mail, Server

GetMail = Mail
End Function

```

Data Type BV/V,V,V

IDL

```

Program 'BinVVVV' Is
    Define Data Parameter
        1 Client    (BV/V,V,V)    In
        1 Mail     (BV/V,V,V)    In Out
        1 Server   (BV/V,V,V)    Out
    End-Define

```

Visual Basic Code

```

Private Function GetMail() As Variant()
    Dim Client() As Variant
    Dim Mail() As Variant
    Dim Server() As Variant

    Dim ClientWrap() As Variant
    Dim MailWrap() As Variant

    Dim ClientValue() As Byte
    Dim MailValue() As Byte

    ReDim Client(3)
    For i = 0 To UBound(Client)
        ReDim ClientWrap(i + 1)
    Next i

```

```
For j = 0 To UBound(ClientWrap)
  ReDim ClientValue(j + 3)
  For k = 0 To UBound(ClientValue)
    ClientValue(k) = i + j + k
  Next k
  ClientWrap(j) = ClientValue
Next j
Client(i) = ClientWrap
Next i

ReDim Mail(7)
For i = 0 To UBound(Mail)
  ReDim MailWrap(i + 2)
  For j = 0 To UBound(MailWrap)
    ReDim MailValue(j + 1)
    For k = 0 To UBound(MailValue)
      MailValue(k) = i + j + k
    Next k
    MailWrap(j) = MailValue
  Next j
  Mail(i) = MailWrap
Next i

obj.BinVVVV Client, Mail, Server

GetMail = Server
End Function
```

Data Type Group

IDL

```
Program 'PGroup1' Is
  Define Data Parameter
    1 Data      In Out
    2 Client    (AV)
    2 Mail      (AV)
  End-Define
```


Visual Basic Code

```

Private Sub GetMail()
    obj.PGroup1_Data.Client = "Caroline"
    obj.PGroup1_Data.Mail = "Susan"

    obj.PGroup1 obj.PGroup1_Data

    if obj.PGroup1_Data.Mail <> "Humpty Dumpty" Then
        MsgBox "Humpty Dumpty is not in"
    end if
End Sub

```

Data Type Group/<nn>

IDL

```

Program 'PGroup2' Is
    Define Data Parameter
        1 Data      (/2) In Out
        2 Client    (AV)
        2 Mail      (AV)
    End-Define

```

Visual Basic Code

```

Private Sub GetMail()
    For i = 0 To UBound(obj.PGroup2_Data)
        obj.PGroup2_Data(i).Client = "Caroline"
        obj.PGroup2_Data(i).Mail = "Susan"
    Next I

    obj.PGroup2 obj.PGroup2_Data

    if obj.PGroup2_Data(0).Mail <> "Humpty Dumpty" Then
        MsgBox "Humpty Dumpty is not in"
    end if
End Sub

```

Data Type Group with Nested Group

IDL

```
Program 'PGroup3' Is
  Define Data Parameter
    1 Data      (/2)  In Out
      2 Client1   (AV)
      2 Mail1    (AV)
      2 SecondLevel (/1:2) In Out
        3 Client2 (AV)
        3 Mail2   (AV)
    1 User      (AV)   In Out
  End-Define
```

Visual Basic Code

```
Private Sub GetMail()
  Dim User As String

  For i = 0 To UBound(obj.PGroup3_Data)
    obj.PGroup3_Data(i).Client1 = "Caroline"
    obj.PGroup3_Data(i).Mail1 = "Susan"
    For j = LBound(obj.PGroup3_Data(i).SecondLevel) To ↵
UBound(obj.PGroup3_Data(i).SecondLevel)
      obj.PGroup3_Data(i).SecondLevel(j).Client2 = "Alice"
      obj.PGroup3_Data(i).SecondLevel(j).Mail2 = "Mary"
    Next j
  Next i

  User = "Minime"

  obj.PGroup3 obj.PGroup3_Data, User

  if obj.PGroup3_Data(0).Mail1 <> "Humpty Dumpty" And _
    obj.PGroup3_Data(0).Mail1.SecondLevel(0).Mail2 = "Humpty Dumpty" Then
    MsgBox "Humpty Dumpty is not in today, but tomorrow"
  end if
End Sub
```

Data Type Group/<nn>,<nn>

IDL

```

Program 'PGroup4' Is
  Define Data Parameter
    1 Data      (/2,3)  In Out
    2 Client    (AV)
    2 Mail      (AV)
  End-Define

```

Visual Basic Code

```

Private Sub GetMail()
  For i = 0 To UBound(obj.PGroup4_Data, 1)
    For j = 0 To UBound(obj.PGroup4_Data, 2)
      obj.PGroup4_Data(i, j).Client = "Caroline"
      obj.PGroup4_Data(i, j).Mail = "Susan"
    Next j
  Next i

  obj.PGroup4 obj.PGroup4_Data

  if obj.PGroup3_Data(0, 0).Mail1 <> "Humpty Dumpty" Then
    MsgBox "Humpty Dumpty is not in today, but tomorrow"
  end if
End Sub

```

Data Type Array Nested in Group

IDL

```

Library 'InpLIB' Is
Program 'PROG' Is
  Define Data Parameter
    1 I          In
    2 inp        (/1:100)
    3 name       (a100)
  End-Define

```

Visual Basic Code

```
Dim Iinpall() As Object
Dim obj As Object
Dim str As String

Set obj = CreateObject("EOL.InpLIB")
obj.ServerAddress = "localhost@RPC/SRV1/CALLNAT"

Iinpall = obj.PROG_i.inp_all

For i = LBound(Iinpall) To UBound(Iinpall)
    Iinpall(i).Name = "test"
Next i

' call the DCOM method
obj.PROG obj.PROG_i

Iinpall = obj.PROG_i.inp_all

For i = LBound(Iinpall) To UBound(Iinpall)
    str = Iinpall(i).Name
Next i
```

Data Type Struct

IDL

```
Struct 'SU' is
    Define Data Parameter
        1 Name (AV)
        1 Age (I2)
    End-Define

Program 'ConstSU' Is
    Define Data Parameter
        1 Client ('SU') In
        1 Mail ('SU') In Out
    End-Define
```

Visual Basic Code

```

Private Function GetMail() As Boolean
    Dim ClientSu As Object ' Client structure
    Dim MailSu As Object ' Mail structure

    Set ClientSu = obj.createStructure_SU
    Set MailSu = obj.createStructure_SU

    ClientSu.Name = "Caroline"
    ClientSu.Age = 21

    MailSu.Name = "Susan"
    MailSu.Age = 24

    obj.ConstSU ClientSu, MailSu

    If MailSu.Name <> "Humpty Dumpty" Then
        GetMail = False
    Else
        GetMail = True
    End If
End Function

```

Data Type Struct/<nn>

IDL

```

Struct 'SU' is
    Define Data Parameter
        1 Name (AV)
        1 Age (I2)
    End-Define

Program 'ConstSunn' Is
    Define Data Parameter
        1 Client ('SU'/3) In
        1 Mail ('SU'/3) In Out
    End-Define

```

Visual Basic Code

```
Private Function GetMail() As Object()  
    Dim ClientSu(2) As Object ' Client structure  
    Dim MailSu(2) As Object ' Mail structure  
  
    For i = 0 To UBound(Client)  
        Set ClientSu(i) = obj.createStructure_SU  
    Next i  
  
    For i = 0 To UBound(MailSu)  
        Set MailSu(i) = obj.createStructure_SU  
    Next i  
  
    ClientSu(0).Name = "Asterix"  
    ClientSu(0).Age = 31  
    ClientSu(1).Name = "Automatix"  
    ClientSu(1).Age = 34  
    ClientSu(2).Name = "Idefix"  
    ClientSu(2).Age = 7  
  
    MailSu(0).Name = "Falbala"  
    MailSu(0).Age = 19  
    MailSu(1).Name = "Gutemine"  
    MailSu(1).Age = 50  
    MailSu(2).Name = "Cleopatra"  
    MailSu(2).Age = 21  
  
    obj.ConstSU ClientSu, MailSu  
    GetMail = MailSu  
End Function
```

Data Type Struct/V

IDL

```
Struct 'SU' is  
    Define Data Parameter  
        1 Name (A40)  
        1 Age (I2)  
    End-Define  
  
Program 'ConstSunn' Is  
    Define Data Parameter  
        1 Client ('SU'/V) In  
        1 Mail ('SU'/V) In Out  
    End-Define
```

Visual Basic Code

```
Private Function GetMail() As Object()  
    Dim ClientSu() As Object ' Client structure  
    Dim MailSu() As Object ' Mail structure  
  
    ReDim ClientSu(3) As Object  
    ReDim MailSu(2) As Object  
  
    For i = 0 To UBound(Client)  
        Set ClientSu(i) = obj.createStructure_SU  
    Next i  
  
    For i = 0 To UBound(MailSu)  
        Set MailSu(i) = obj.createStructure_SU  
    Next i  
  
    ClientSu(0).Name = "Asterix"  
    ClientSu(0).Age = 31  
    ClientSu(1).Name = "Automatix"  
    ClientSu(1).Age = 34  
    ClientSu(2).Name = "Idefix"  
    ClientSu(2).Age = 7  
    ClientSu(3).Name = "Obelix"  
    ClientSu(3).Age = 30  
  
    MailSu(0).Name = "Falbala"  
    MailSu(0).Age = 19  
    MailSu(1).Name = "Gutemine"  
    MailSu(1).Age = 50  
    MailSu(2).Name = "Cleopatra"  
    MailSu(2).Age = 21  
  
    obj.ConstSU ClientSu, MailSu  
    GetMail = MailSu  
End Function
```

Data Type Struct/V,V

IDL

```
Struct 'SU' is
  Define Data Parameter
    1 Name (A20)
    1 Age (I2)
  End-Define

Program 'ConstSUvv' Is
  Define Data Parameter
    1 Client ('SU'/V,V) In
    1 Mail ('SU'/V,V) In Out
  End-Define
```

Visual Basic Code

```
Private Function GetMail() As Variant()
  Dim ClientSu() As Variant ' Client structure
  Dim MailSu() As Object ' Mail structure
  Dim i as Integer
  Dim j as Integer

  ' helper variable
  Dim ClientSu1() As Object
  ReDim ClientSu(3)
  ReDim ClientSu1(6)

  For i = 0 To UBound(ClientSu)
    For j = 0 To UBound(ClientSu1)
      Set ClientSu1(j) = obj.createStructure_SU()
      ClientSu1(j).Name = "ClientName_" & Str(j)
      ClientSu1(j).Age = j
    Next
    ClientSu(i) = ClientSu1
  Next

  ' helper variable
  Dim MailSu1() As Object
  ReDim MailSu(5)
  ReDim MailSu1(2)
  For i = 0 To UBound(MailSu)
    For j = 0 To UBound(MailSu1)
      Set MailSu1(j) = obj.createStructure_SU()
      MailSu1(j).Name = "MailName_" & Str(j)
      MailSu1(j).Age = j+2
    Next
  Next
```



```

        Next
        MailSu(i) = MailSu1
    Next
    obj.ConstSUvv ClientSu, MailSu
    GetMail = MailSu
End Function

```

Data Type Struct/V,V,V

IDL

```

Struct 'SU' is
    Define Data Parameter
        1 Name (A20)
        1 Age (I2)
    End-Define

```

```

Program 'ConstSUvvv' Is
    Define Data Parameter
        1 Client ('SU2'/V,V,V) In
        1 Mail ('SU2'/V,V,V) In Out
    End-Define

```

Visual Basic Code

```

Private Function GetMail() As Variant()
    Dim ClientSu() As Variant ' Client structure
    Dim MailSu() As Object ' Mail structure

```

```

' helper variable
Dim ClientSU1() As Variant
Dim ClientSU2() As Object
ReDim ClientSu(3)
ReDim ClientSu1(6)
ReDim ClientSu2(2)

```

```

For i = 0 To UBound(ClientSu)
    For j = 0 To UBound(ClientSu1)
        For k = 0 To UBound(ClientSu2)
            Set ClientSu2(k) = obj.createStructure_SU()
            ClientSu2(k).Name = "ClientName_" & Str(k)
            ClientSu2(k).Age = k
        Next
        ClientSu1(j) = ClientSu2
    Next
Next

```

```
ClientSu(i) = ClientSu1  
Next
```

```
' helper variable  
Dim MailSu1() As Variant  
Dim MailSu2() As Object  
ReDim MailSu(5)  
ReDim MailSu1(2)  
ReDim MailSu2(7)
```

```
For i = 0 To UBound(MailSu)  
    For j = 0 To UBound(MailSu1)  
        For k = 0 To UBound(MailSu2)  
            Set MailSu2(k) = obj.createStructure_SU()  
            MailSu2(k).Name = "MailName_" & Str(k)  
            MailSu2(k).Age = k+2  
        Next  
        MailSu1(j) = MailSu2  
    Next  
    MailSu(i) = MailSu1  
Next
```

```
obj.ConstSU ClientSu, MailSu  
GetMail = MailSu  
End Function
```

16 Using Arrays of Variable Sizes with the DCOM Wrapper

- Description 148
- Methods for Arrays of Variable Sizes 148
- Group Definition for Arrays of Variable Sizes 150

Description

The following describes an array without fixed size and with strings of variable length. It is sent from client to server and vice versa. The number of strings sent from client to server and from server to client can differ.

Software AG IDL File

```
Library 'Arrays' Is
  Program 'FirstArray'
    1 Names (AV / V) In Out
```

Visual Basic Example

```
Dim names As String
Redim mynames(3)

Set obj = CreateObject("EOL.ArrayExample")

obj.FirstArray mynames
```

To retrieve the list of mynames, first get the boundaries of the returning array; then you can access it.

For more information, see the Visual Basic documentation.

Methods for Arrays of Variable Sizes

Attributes defined inside a group in the Software AG IDL file are accessed via properties (as in the previous version). To use arrays of variable sizes, two methods were added: `get` and `redim`.

Software AG IDL File

```
Library 'Arrays' Is
  Program 'FirstArray'
    1 PhoneEntry
    2 Names (AV) In Out
    2 PhoneNumber (AV/V) In Out
```

Array of a Group with Variable Size on Level 1 in the IDL File

```

redim_<program name>_<name>( LONG newSize,
                             LONG whichDimension,
                             LONG index1,
                             LONG index2,
                             LONG index3 )

get_<program name>_Bounds (
    LONG* lowerBoundary,
    LONG* upperBoundary,
    LONG whichDimension,
    LONG index1,
    LONG index2,
    LONG index3 )

LONG get_<program name>_LowerBound (
    LONG whichDimension,
    LONG index1,
    LONG index2,
    LONG index3 )

LONG get_<program name>_UpperBound (
    LONG whichDimension,
    LONG index1,
    LONG index2,
    LONG index3 )

```

Otherwise

```

redim_<name>( LONG newSize,
              LONG whichDimension,
              LONG index1,
              LONG index2,
              LONG index3 )

get_<name>_Bounds(
    LONG* lowerBoundary,
    LONG* upperBoundary,
    LONG whichDimension,
    LONG index1,
    LONG index2,
    LONG index3 )

LONG get_<name>_LowerBound (

```

```
LONG whichDimension,  
LONG index1,  
LONG index2,  
LONG index3 )  
  
LONG get_<name>_UpperBound (   
LONG whichDimension,  
LONG index1,  
LONG index2,  
LONG index3 )
```

Use `redim` to increase or reduce the number of elements in an array. In a multi-dimensional array you can use the indices to modify a subarray.

Group Definition for Arrays of Variable Sizes

```
Library 'Arrays' Is  
Program 'ThirdArray'  
1 User (V)  
2 Names (AV)  
2 Uid (AV)
```

Groups with arrays of variable sizes are handled like normal groups using the methods `get` and `redim`:

```
redim_[<program name>_]<groupname>
```

and

```
get_[<program name>_]<groupname>_bounds.
```

17 Using Structures with the DCOM Wrapper

▪ Overview	152
▪ Software AG IDL File	152
▪ Using Structures	156

The DCOM Wrapper maps a structure definition to an object inside an automation object. The structure can be created and modified independently of program calls, e.g., a structure object can be created once and can be used in one or more programs several times - it is always the same object with the same information. It is also possible to retrieve a structure via a program call and use it in another program.

Overview

A structure definition has always a name like a program name. The difference is that a structure will be defined with the keyword `STRUCT`. Furthermore a structure cannot be called like a program. A structure represents only a data object to be used with the programs. A structure has more the character of a group but unlike a group a structure will not be created automatically by the DCOM Wrapper object. The user has to create a structure with the corresponding functions.

- If a structure has embedded structures, first create the parent structure and then the embedded structures.
- If a structure has embedded groups, the embedded groups are created automatically when the parent structure. is created.
- If a group has embedded structures, the embedded structures must be created by the user. The parent group self is created by DCOM Wrapper object.

Software AG IDL File

A Simple Structure Example

```
LIBRARY 'STRUCTLIB' is

STRUCT 'User' Is
DEFINE DATA PARAMETER
  1 UserId      (AV)
  1 Password    (AV)
END-DEFINE

PROGRAM 'Prog01' Is
DEFINE DATA PARAMETER
  1 ID          ('User')
  1 something   (AV)
END-DEFINE

PROGRAM 'Prog02' Is
DEFINE DATA PARAMETER
  1 ID          ('User')
```



```
1 somewhere (AV)
END-DEFINE
```

Example for Visual Basic Version 6

```
'first create an instance of the DCOM Wrapper Object
Dim obj As New STRUCTLIB
'Declares a variable for our User structure
Dim userobj As Object
'Declares a variable for the second parameter
Dim something as String

'create the 'User' structure
Set userobj = obj.createStructure_User
'now you can access the structure items
userobj.UserId = "userid"
userobj.Password = "password"

something = "Hello"

'call the DCOM object
obj.Prog01 userobj, something
'the same userobj could now used by obj.Prog02. Please consider that
'obj.Prog01 may have changed the data if you reuse userobj.

'get the response from the server and show the message a messagebox
MsgBox "User ID = " & userobj.UserId
```

A Simple Structure Example

Structures can also be defined in a more complex manner. This example shows a definition of a structure in a structure. In this example the customer can have multiple addresses and could have bought multiple products. The following example may not make sense, it is only an example to show the handling of complex structures and embedded structures.

```
LIBRARY 'CUSTLIB' is

STRUCT 'AddressStruct' Is
DEFINE DATA PARAMETER
1 Street (A255)
1 City (A255)
1 Postal_Code (A32)
END-DEFINE

STRUCT 'CustomerStruct' Is
DEFINE DATA PARAMETER
1 Name (A255)
1 Address ('AddressStruct'/V)
END-DEFINE
```

```
STRUCT 'SalesStruct' Is
DEFINE DATA PARAMETER
  1 ProductName (A255)
  1 ProductID   (PU20)
  1 SalesDate   (D)
  1 SalesVolume (N12.2)
END-DEFINE

PROGRAM 'CustomerInformation' is
DEFINE DATA PARAMETER
  1 Customer    ('CustomerStruct')
  1 SalesData   ('SalesStruct'/V)
END-DEFINE
```

Example for Visual Basic Version 6

```
'first create an instance of the DCOM Wrapper Object
Dim obj As New CUSTLIB
'Declares an array of variables for our sales structure
Dim salesobj() As Object
'Declares a variable for our customer structure
Dim custobj As Object
'Declares a variable for our address structure
Dim Addressobj As Object
'temp strings
Dim addressdata As String
Dim salesdata As String
Dim i As Integer

'create the 'Customer' structure
Set custobj = obj.createStructure_CustomerStruct
'redim the Customer structure to the size 2 in the first dimension.
custobj.redim_address 2, 1, 0, 0, 0
'set the name of the customer
custobj.Name = "Customer A"
'create a new address structure
Set Addressobj = obj.createStructure_AddressStruct
'fill the data items of this structure
Addressobj.street = "back street"
Addressobj.city = "back town"
Addressobj.postal_code = "12345"
'assign the address structure to the first array item of the
'Customer structure.
custobj.address_indexAccess(0) = Addressobj

'create the second 'Customer' structure
Set Addressobj = obj.createStructure_AddressStruct
'fill the data items of this structure, too
Addressobj.street = "front street"
Addressobj.city = "front town"
```

```

Addressobj.postal_code = "54321"
'assign the address structure to the second array item of the
'Customer structure.
custobj.address_indexAccess(1) = Addressobj

'Redim the sales array object, the salesobj has now one
'array item
ReDim salesobj(0)
'create a new sales structure
Set salesobj(0) = obj.createStructure_SalesStruct
'fill the data items of this structure
salesobj(0).ProductName = "phone"
salesobj(0).productid = "12547"
salesobj(0).salesdate = Now
salesobj(0).salesvolume = "99.91"

'call the DCOM object with the two parameters.
obj.CustomerInformation custobj, salesobj

'now show the received data
addressdata = ""
addressdata = custobj.Name & vbCrLf

'please note that you need to use the DCOM Wrapper special function
'to access the embedded structures.
For i = 0 To custobj.get_address_UpperBound(1, 0)
    'get a pointer to the address data
    Set Addressobj = custobj.address_indexAccess(i)
    'pick the data from the address structure
    addressdata = addressdata & Addressobj.street & ";" & _
    Addressobj.city & vbCrLf
Next

'please note that you don't have a special function to access
'a structure array for the data on the first level.
salesdata = addressdata & vbCrLf
' show the other sales
For i = 0 To UBound(salesobj)
'pick the data from the address structure
    salesdata = salesdata & salesobj(i).ProductName & ";" & _
    CLng(salesobj(i).salesvolume) & vbCrLf
Next

'now show data in a messagebox
MsgBox (salesdata)

```

Using Structures

Creating New Instances of a Structure

The DCOM Wrapper creates a function for each structure definition. The function is always a member of the library object and has always the same naming template. It begins with the keyword *createStructure_* with a following structure name. This function returns a IDispatch object pointer to a new valid structure data object.

```
STDMETHODIMP createStructure_<structure name>( IDispatch ** );
```

The method `createStructure_<structure name>` creates a new instance of the named original structure, a pointer to this structure is returned.

Example

IDL File

```
LIBRARY 'EXAMPLELIB' is
STRUCT 'SData' Is
DEFINE DATA PARAMETER
    1 Name      (A255)
    1 Address   (A255)
END-DEFINE
.
.
.
.
```

Visual Basic Version 6 Source Code

```
Dim sdata As Object
Dim obj As New EXAMPLELIB
set sdata = obj.createStructure_SData
```

.NET C# Source Code

```
IStructure_SData sdata;  
EXAMPLELIBClass obj = new EXAMPLELIBClass();  
sdata = (IStructure_SData)obj.createStructure_SData();
```


18 Writing Applications with the DCOM Wrapper

▪ Tracing	160
▪ Using Internationalization with the DCOM Wrapper	160
▪ Setting Transport Methods for the DCOM Wrapper	161

Tracing

There are several possibilities to trace the DCOM Wrapper. For details, see *Tracing webMethods EntireX* in the platform-specific administration documentation.

Using Internationalization with the DCOM Wrapper

It is assumed that you have read the document *Internationalization with EntireX* and are familiar with the various internationalization approaches described there.

The DCOM Wrapper uses by default the Windows ANSI codepage (CP_ACP) to convert the Unicode (UTF-16) representation within BSTRINGS to the multibyte or single-byte encoding sent to/received from the broker. This codepage is also transferred as part of the locale string to tell the broker the encoding of the data if communicating with a broker version 7.2.x and above.

If you want to adapt the Windows ANSI codepage refer to the Regional Settings in the Windows Control Panel and your Windows documentation.

The following applies to the property codepage:

- A DCOM Wrapper programmer can override the default Windows ANSI codepage the DCOM Wrapper uses. Use the Windows codepage number in numerical form to specify a different one, see the property Codepage in the section *Standard Wrapper Properties*. If a codepage is provided, it is transferred as part of the locale string to the Broker for all Broker versions. Use CP_ACP to send the default Windows ANSI codepage to the Broker.
- The Windows codepage used by the DCOM Wrapper must also be a supported codepage of the broker depending on the internationalization approach. See *Locale String Mapping* in the internationalization documentation for information on how the broker derives the codepage from the locale string.

If communicating with a Broker version 7.1.x and below and if no locale string is provided by the DCOM Wrapper programmer, an administrator can force a locale string to be sent with the environment variable `ERX_CODEPAGE`.

When setting the codepage with the environment variable `ERX_CODEPAGE`:

- The `ERX_CODEPAGE` environment variable is ignored if the application programmer has already provided a codepage.
- The value of the `ERX_CODEPAGE` environment variable must be the name of the system's default codepage. Under Windows, simply apply the value "LOCAL" to specify the default Windows ANSI codepage.

- The codepage specified must be one that is supported by the Broker, depending on the Broker's internationalization approach. See *Locale String Mapping* in the internationalization documentation for information on how the broker derives the codepage from the locale string.
- Before starting the application, set the locale string with the environment variable `ERX_CODEPAGE`.

Example:

```
ERX_CODEPAGE=LOCAL
```

Setting Transport Methods for the DCOM Wrapper

The procedure for setting the transport methods and timeout for the EntireX DCOM Wrapper is the same as for the Broker stubs.

See *Transport Methods for Broker Stubs* in the platform-specific broker stub administration documentation.

19 Microsoft Visual Studio Wizard for EntireX DCOM Wrapper

- Creating a new DCOM Wrapper Project 164

The Microsoft Visual Studio Wizard for EntireX DCOM Wrapper is a Software AG add-in that enables Microsoft Visual Studio 2010 to generate Visual C# .NET and Visual Basic .NET Windows applications that use EntireX DCOM Wrapper objects. The Visual Studio Wizard creates a DCOM Wrapper object from a Software AG IDL file and adds a reference to the object into the new Windows application.



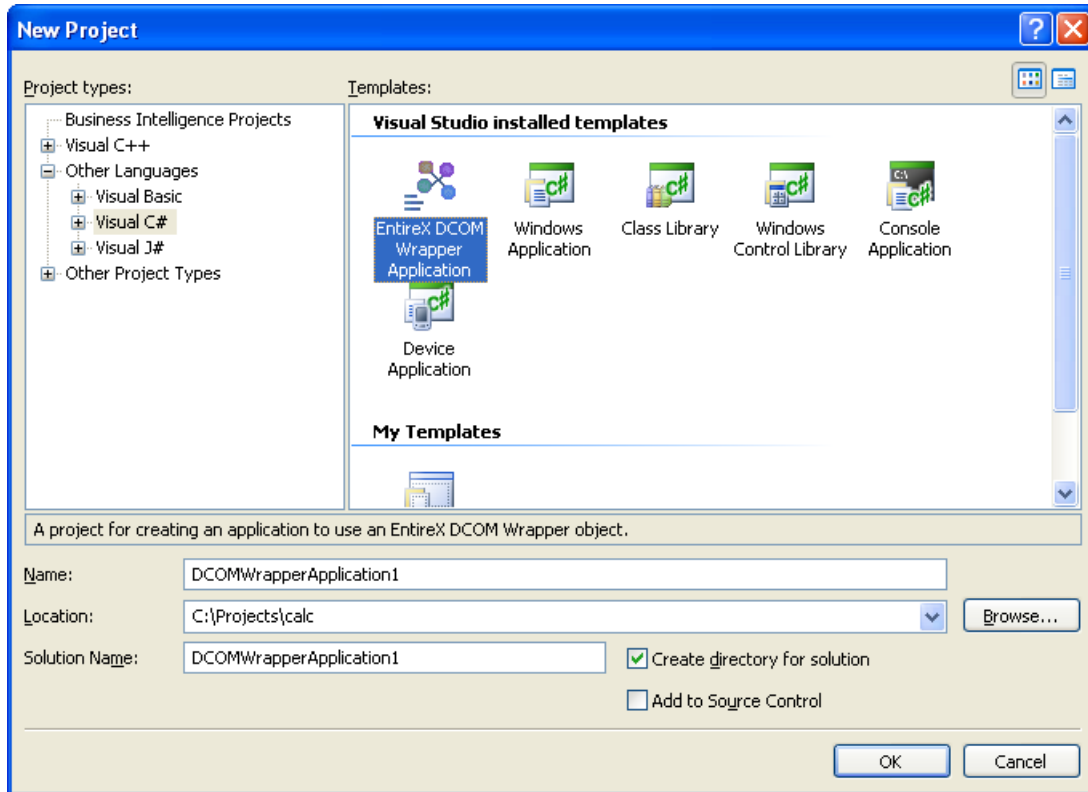
Notes:

1. The dialog boxes might differ depending on your active Visual Studio settings.
2. Generation of a DCOM Wrapper proxy is not supported in the Visual Studio Wizard.

Creating a new DCOM Wrapper Project

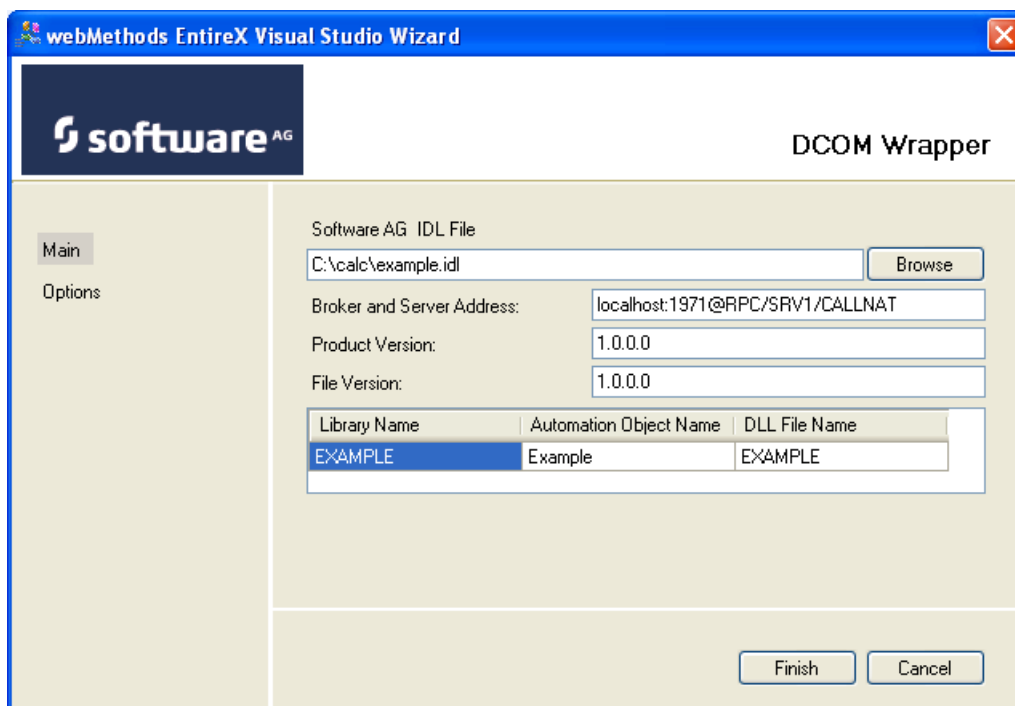
▶ **To create a new EntireX DCOM Wrapper Windows Application**

- 1 Select the **File > New > Project** menu.
- 2 Choose your preferred programming language to create a new project. The Visual Studio Wizard is available for Visual C# and Visual Basic.
- 3 Select the Visual Studio Wizard icon, enter the name of the project and choose the project location.



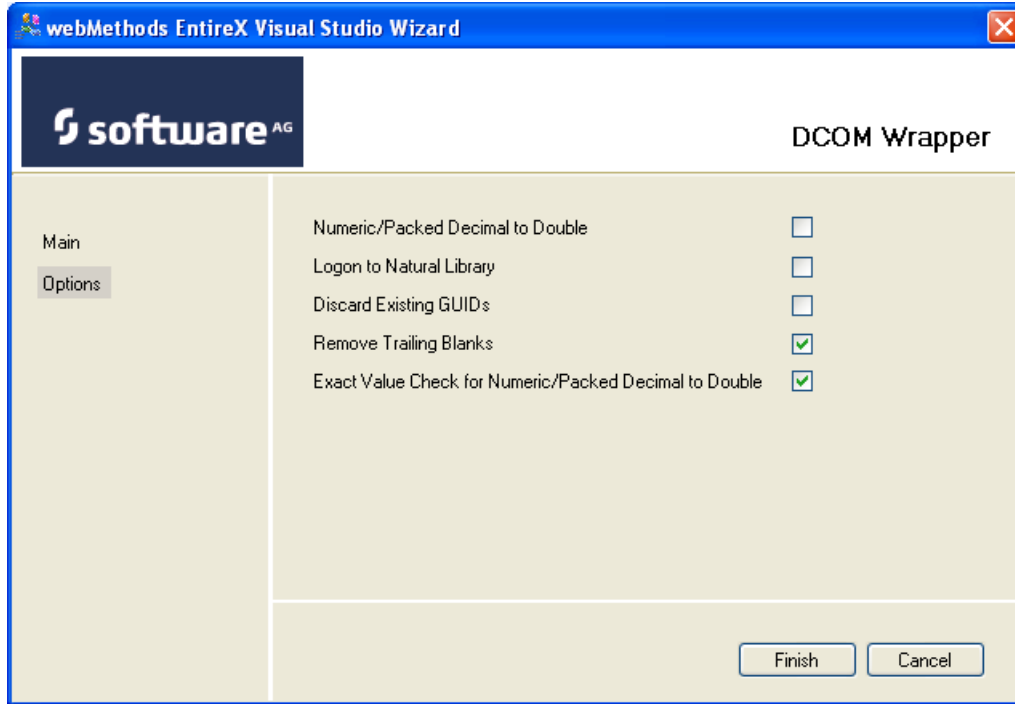
- 4 Choose **OK**.

The Visual Studio Wizard starts and displays the main input panel.



- 5 Specify the Software AG IDL file.
The Software AG IDL file is mandatory in this dialog and the file you specify must be a valid Software AG IDL file.
- 6 Enter the broker and server address.
The broker address is by default set to `localhost:1971`. The server address (`class/server/service`) is by default set to `RPC/SRV1/CALLNAT`. Please enter the required address of the broker and server for your application. Please use always the following notation for the broker and server address.
`<broker address>[:port]@<class>/<server>/<service>`
The port is optional if the broker runs on the default port 1971.
- 7 You can specify the product version and file version. This version information will be displayed in the properties of the generated DCOM DLL.
- 8 The library name is read from the library statement from the IDL file. The property is a read only property in this dialog and cannot be changed.
- 9 The automation name is read from the alias statement of the library. If the alias statement is missing, the automation name is `EOL.<library name>`. You can change this property if required, and it will be used while generating the DCOM Wrapper object. Changes of the automation name in this dialog have no effect on the IDL file; they will not be saved into the IDL file.
- 10 The DLL file name is set by default to the library name. This value will be used while generating the DCOM Wrapper object and sets the output file name. You can set this value to generate a DLL with a name other than the library name. Please use this value without the extension `.dll`. The extension will be added automatically while generating the DCOM Wrapper DLL.
- 11 If you wish to specify further settings, choose the **Options** input panel on the left navigation panel, otherwise proceed to the next step.

The **Options** panel provides additional settings for generating the DCOM Wrapper object. See the [Setting DCOM Wrapper Properties](#) for a detailed description of these properties.



- 12 Choose **Finish** to start the generation of the DCOM Wrapper object. A process window will be displayed while generating the DCOM Wrapper object. When the generation process completes, a simple Visual Basic or Visual C# solution is available and ready to use.

The generated Visual Basic or Visual C# solution contains the reference to the new DCOM Wrapper object. If the Software AG IDL file has more than one library object, only the first library object will be shown in the example implementation, but all library objects will be added to the references of the solution.

