webMethods EntireX

EntireX Broker ActiveX Control

Version 10.9

April 2023

Document ID: EXX-EEXXACI-109-20230403ACTX

Table of Contents

Preface v
1 About this Documentation 1
    Document Conventions 2
    Online Information and Support 2
    Data Protection 3
2 Broker ActiveX Control Introduction 5
    Broker ACI 6
    Transaction Objects 6
3 Writing Applications - Broker ActiveX Control 7
    Calling a Broker Function 8
    Viewing the Type Library 10
    Adding the Broker ActiveX Control Component to Visual Studio 11
    Using the Property Pages 14
4 Broker ActiveX Control with Visual Basic 17
    Step 1: Instantiate EntireX Broker ActiveX Control 18
    Step 2: Instantiate the Transaction Object 20
    Step 3: Call Methods 20
    Step 4: Access the Returned Data 21
    Step 5: Cleanup Resources 24
    Step 6: Error Handling in Transaction Object Methods 24
    Examples: Writing an ACI Client and Server with Broker ActiveX Control 24
5 Using Broker ActiveX Control with Active Server Pages 27
    Prerequisites 28
    Designing a Web Page with ASP and Broker ActiveX Control 28
    Using Broker ActiveX Control in Multiple Pages 30
6 Using Broker ActiveX Control with .NET 31
    Using Broker ActiveX Control with Visual Studio .NET 32
    A Small Visual Basic .NET Example 32
7 Transaction Objects in Broker ActiveX Control 33
    Advantages of Transaction Objects 34
    Calling the Transaction Object Editor 34
    Managing TOR Files 36
    Defining Methods 39
    Specifying Connection Information 44
    Defining Custom Data Types 46
8 Calling Broker ActiveX Control Remotely 49
    Setting up the Server Environment 50
    Setting up the Client Environment 55
    Testing the Connection 57
9 Reference 61
    Methods 62
    Properties 63

Preface

Broker ActiveX Control allows GUI application developers to use an ActiveX-based interface to access EntireX Broker. It can be used within ActiveX containers, such as Visual Basic, PowerBuilder, Delphi, Microsoft Excel, Microsoft Word.

Broker ActiveX Control Introduction	Broker ActiveX Control provides a programmatic interface to COM-enabled programming environments. It has two types of operation: the Broker ACI and transaction objects. Broker ActiveX Control enables you to create EntireX ACI clients and EntireX ACI servers.
Writing Applications - Broker ActiveX Control	Topics include calling a Broker function; viewing the type library; using property pages.
Broker ActiveX Control with Visual Basic	Visual Basic is used here as an example of a development environment in which applications using Broker ActiveX Control can work. Broker ActiveX Control can be used by any programming language or programming environment that can act as a container for ActiveX controls.
Using Broker ActiveX Control with Active Server Pages	Microsoft's Active Server Page (ASP) is an HTML page that includes one or more scripts and reusable ActiveX server components to create dynamic Web pages. The scripts and ActiveX components are processed on a Microsoft Web server before the page is sent to the user.
Using Broker ActiveX Control with .NET	How to use Broker ActiveX Control with Visual Studio .NET. An example is provided.
Transaction Objects in Broker ActiveX Control	Transaction objects (TOs) in Broker ActiveX Control are selections of logical methods that are stored in a transaction object repository (TOR). These logical methods contain all the connection and interface details necessary to communicate with the Broker.
Calling Broker ActiveX Control Remotely	You can call Broker ActiveX Control remotely if you use it as an automation server. This means you can use the Broker component from a separate process - either on the same machine or on another machine in the network.
Reference	Methods and properties of Broker ActiveX Control.

1 About this Documentation

- Document Conventions 2
- Online Information and Support 2
- Data Protection 3

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
`Monospace font`	Identifies service names and locations in the format *folder.subfolder.service*, APIs, Java classes, methods, properties.
Italic	Identifies: Variables for which you must supply values specific to your own situation or environment. New terms the first time they occur in the text. References to other documentation sources.
`Monospace font`	Identifies: Text you must type in. Messages displayed by the system. Program code.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information and Support

Product Documentation

You can find the product documentation on our documentation website at **https://documentation.softwareag.com**.

In addition, you can also access the cloud product documentation via **https://www.softwareag.cloud**. Navigate to the desired product and then, depending on your solution, go to “Developer Center”, “User Center” or “Documentation”.

Product Training

You can find helpful product training material on our Learning Portal at **https://knowledge.softwareag.com**.

Tech Community

You can collaborate with Software AG experts on our Tech Community website at **https://tech-community.softwareag.com**. From here you can, for example:

- Browse through our vast knowledge base.
- Ask questions and find answers in our discussion forums.
- Get the latest Software AG news and announcements.
- Explore our communities.
- Go to our public GitHub and Docker repositories at **https://github.com/softwareag** and **https://hub.docker.com/publishers/softwareag** and discover additional Software AG resources.

Product Support

Support for Software AG products is provided to licensed customers via our Empower Portal at **https://empower.softwareag.com**. Many services on this portal require that you have an account. If you do not yet have one, you can request it at **https://empower.softwareag.com/register**. Once you have an account, you can, for example:

- Download products, updates and fixes.
- Search the Knowledge Center for technical information and tips.
- Subscribe to early warnings and critical alerts.
- Open and update support incidents.
- Add product feature requests.

Data Protection

Software AG products provide functionality with respect to processing of personal data according to the EU General Data Protection Regulation (GDPR). Where applicable, appropriate steps are documented in the respective administration documentation.

2 Broker ActiveX Control Introduction

- Broker ACI 6
- Transaction Objects 6

Broker ActiveX Control provides a programmatic interface to COM-enabled programming environments. It has two types of operation: the Broker ACI and transaction objects. Broker ActiveX Control enables the user to create EntireX ACI clients and EntireX ACI servers.

Broker ACI

The Broker ACI provides a simple automation API that is one-to-one compatible with the published EntireX Broker ACI. It provides Broker ActiveX Control properties and corresponding property pages for the control parameters detailed in the Broker ACI fields. This API is conceptually compatible with current Broker programming practices. Further, the Broker ActiveX Control programmer can count on programmatic behavior consistent with programming the Broker API directly, such as non-blocking calls and polling for completion.

Transaction Objects

Broker ActiveX Control generates ActiveX automation server interfaces dynamically at runtime from files in the Transaction Object Repository (TOR).

Broker ActiveX Control transaction objects provide a dictionary subsystem and user interface that will allow the EntireX Broker developer to define a dynamic IDispatch interface. This interface allows received data to be accessed with the traditional automation methodology.

The transaction object definition of a method also includes parsing up the SEND and RECEIVE buffers of a Broker message into parameters and return properties respectively. The transaction objects are loaded at runtime and the ActiveX container can then call the methods of that transaction object to send/receive data.

See ***Transaction Objects in Broker ActiveX Control*** for more information.

3 Writing Applications - Broker ActiveX Control

- Calling a Broker Function 8
- Viewing the Type Library 10
- Adding the Broker ActiveX Control Component to Visual Studio 11
- Using the Property Pages 14

Calling a Broker Function

Setting the Broker ActiveX Properties

You can set the Broker ActiveX properties either in the program or in the property pages. See *Properties*.

Specifying the Send Parameters

Before executing a `send` function, specify the send parameters with the method `SetSendDataLong(String bsData, Long DataLen)` or `SetSendData(String bsData, Short DataLen)`.

This method sets only the send buffer.

The first parameter specifies the buffer that has to be sent to the server. The second parameter specifies the number of bytes to be transferred.

The following rules apply to the `SetSendData` method:

- The `DataLen` bytes of the string `bsData` are copied to the internal send buffer.
- A byte copy is performed (not a string character copy), which means that the string `bsData` can contain zero bytes.
- The function `BOOL SetSendData( String bsData, Short DataLen )` can be used if the send buffer is smaller than 32 KB.

Calling the Broker Function

- Set the required properties.
- When you use the `send` function, use the method `SetSendData` to set up the send buffer.
- When you use the `receive` function, use the property `ReceiveBufferSize` to set up the size of the internal receive buffer.
- Use the static automation method to call the Broker functions:

```
BOOL InvokeBrokerFunction()
```

This method executes the Broker function defined by the current value of the property `Function`. Depending on the function, the required Broker parameters are taken from the current values of the corresponding properties.

If the Broker call is successful:

- The function returns TRUE.
- The `ErrorCode` property is set to '00000000' and the `ErrorMsg` property is empty.

If the Broker call is a `Send` or `Receive` function, this call may also update the `ConvID` property.

If the Broker call is a `Receive` function and asterisks were specified for ServerClass, ServerName and Service, the call updates the `ServerClass`, `ServerName` and `Service` properties.

If the Broker call is a `Receive` or `Send` with implicit `Receive` (Wait > 0), the number of bytes received is stored in the property `ReturnDataLength` and the returned data can be retrieved with the `GetReceiveData` method.

If the Broker call fails:

- The function returns FALSE.
- The `ErrorCode` and `ErrorMsg` properties contain the corresponding error reason.

The error code has two parts:

- error class (first four digits), which provides information for the application on how to react to the returned error, and
- error number (last four digits), which indicates the reason for the error.

The `GetErrorText` method is still available and returns the value of the `ErrorMsg` property.

For more information see *Error Messages and Codes*.

Getting the Contents of the Receive Buffer

If a `Receive` function was executed, the receive buffer can be retrieved with the function

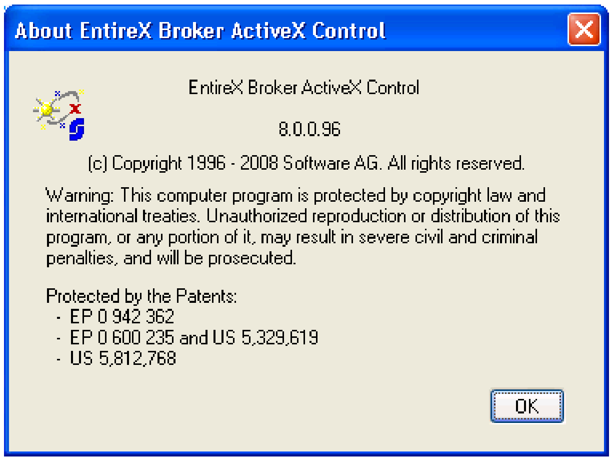
```
STRING GetReceiveData()
```

AboutBox

The `AboutBox` method is used to show the version of Broker ActiveX Control.

A message box will be displayed containing the **About** information.

```
AboutBox ()
```



EntireX Broker ActiveX Control

8.0.0.96

(c) Copyright 1996 - 2008 Software AG. All rights reserved.

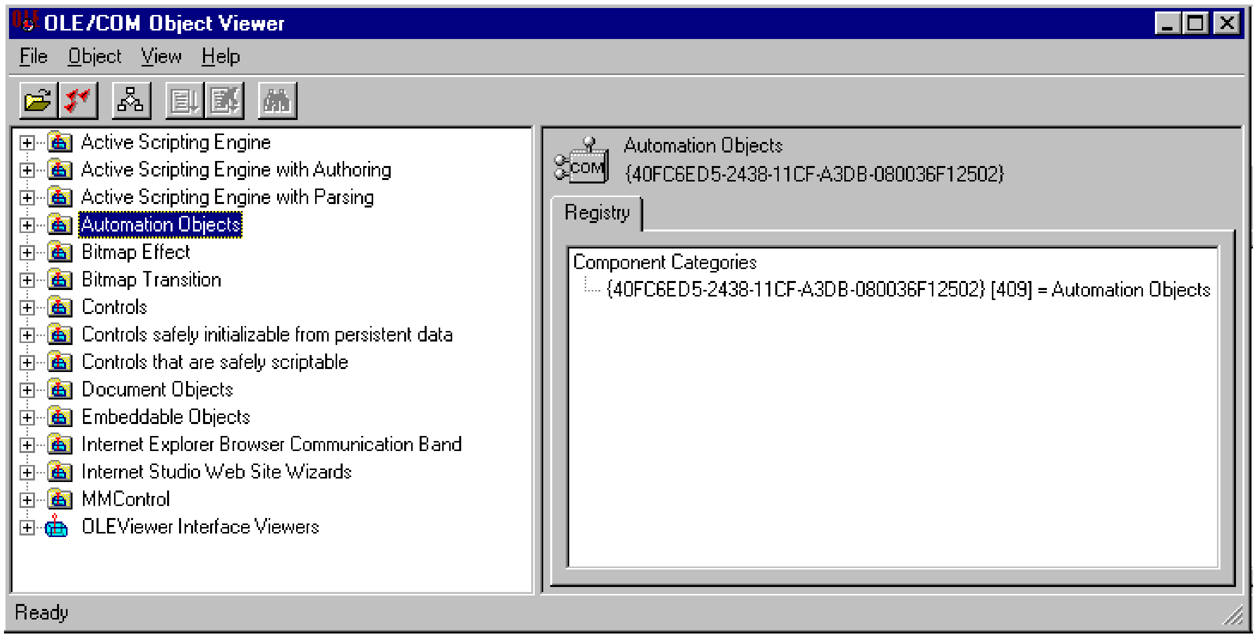
Warning: This computer program is protected by copyright law and international treaties. Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted.

Protected by the Patents:
- EP 0 942 362
- EP 0 600 235 and US 5,329,619
- US 5,812,768

Viewing the Type Library

To view the Type Library of Broker ActiveX Control

- Use the OLE/COM Object Viewer (choose **EntireX Broker ActiveX Control** and choose **View Type Information**).



Active Scripting Engine
Active Scripting Engine with Authoring
Active Scripting Engine with Parsing
Automation Objects
Bitmap Effect
Bitmap Transition
Controls
Controls safely initializable from persistent data
Controls that are safely scriptable
Document Objects
Embeddable Objects
Internet Explorer Browser Communication Band
Internet Studio Web Site Wizards
MMControl
OLEViewer Interface Viewers

Automation Objects
{40FC6ED5-2438-11CF-A3DB-080036F12502}

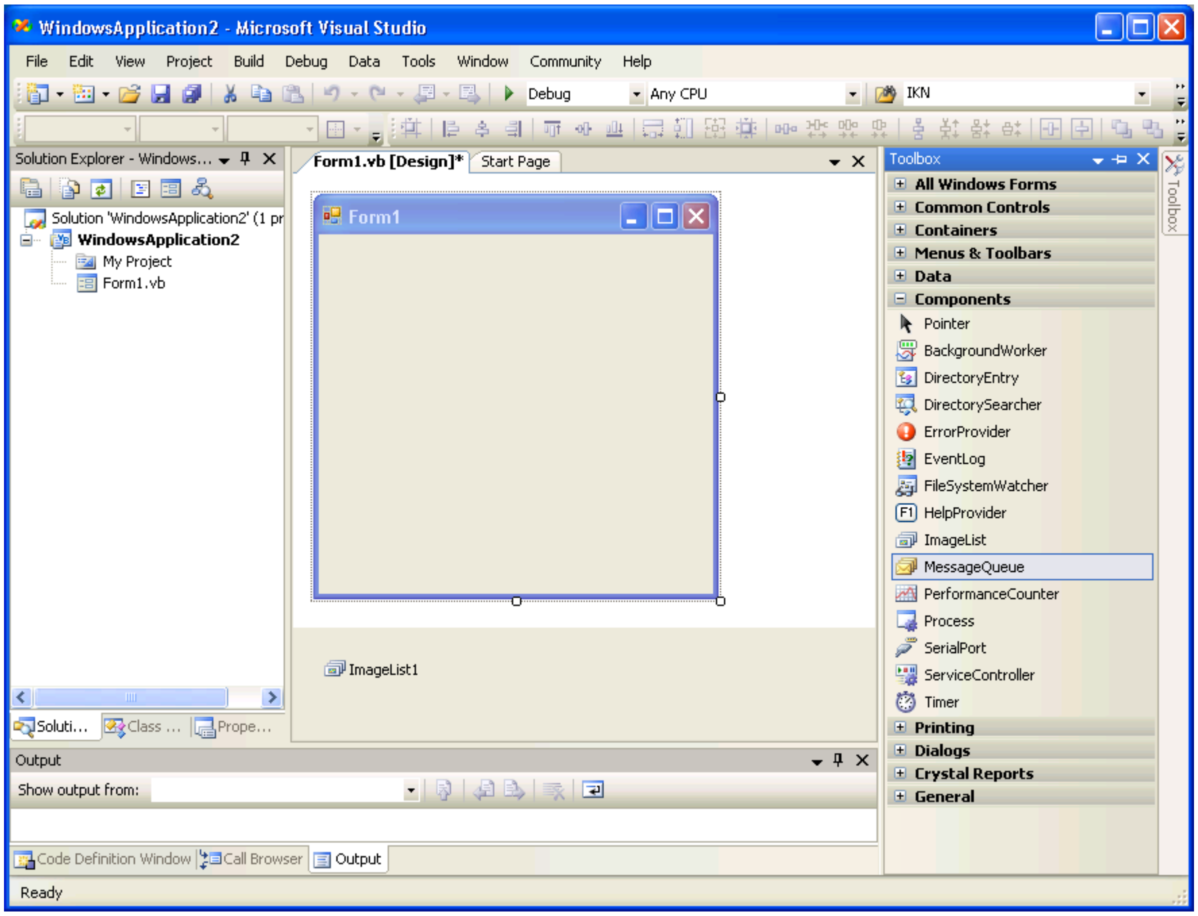
Component Categories
{40FC6ED5-2438-11CF-A3DB-080036F12502} [409] = Automation Objects

To do this with Visual Basic, see *Using Broker ActiveX Control as an Automation Server*.

Adding the Broker ActiveX Control Component to Visual Studio

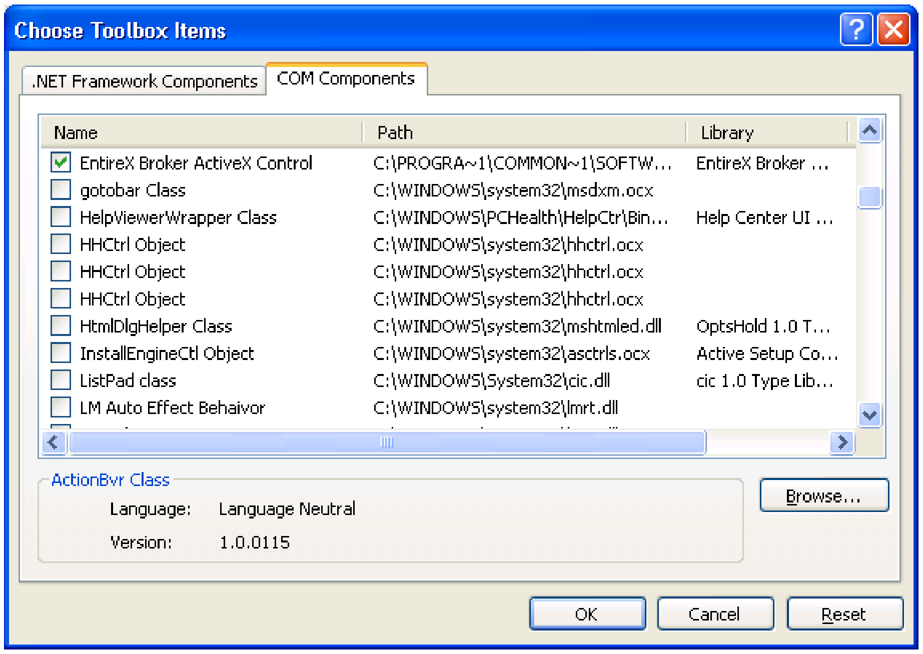
To add the Broker ActiveX Control component to Visual Studio

1 In Visual Studio, choose **Toolbox > Components**.



2 From the context menu, choose **Choose Item**.

3 In the **Choose Toolbox Items** dialog under **COM Components**, check "EntireX Broker ActiveX Control".


Choose Toolbox Items

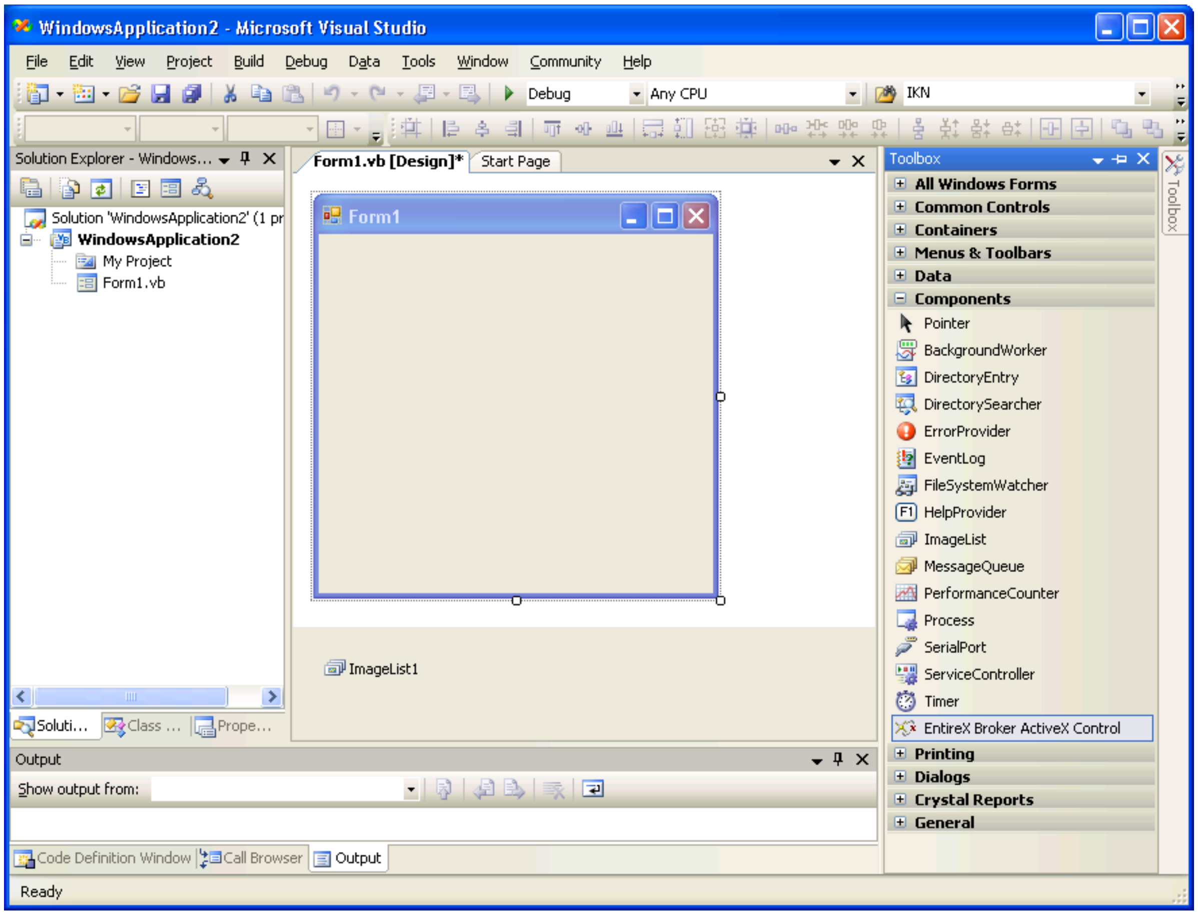
.NET Framework Components | COM Components

Name	Path	Library
✓ EntireX Broker ActiveX Control	C:\PROGRA~1\COMMON~1\SOFTW...	EntireX Broker ...
gotobar Class	C:\WINDOWS\system32\msdxm.ocx	
HelpViewerWrapper Class	C:\WINDOWS\PCHealth\HelpCtr\Bin...	Help Center UI ...
HHCtrl Object	C:\WINDOWS\system32\hhctrl.ocx	
HHCtrl Object	C:\WINDOWS\system32\hhctrl.ocx	
HHCtrl Object	C:\WINDOWS\system32\hhctrl.ocx	
HtmlDlgHelper Class	C:\WINDOWS\system32\mshtmled.dll	OptsHold 1.0 T...
InstallEngineCtl Object	C:\WINDOWS\system32\asctrls.ocx	Active Setup Co...
ListPad class	C:\WINDOWS\System32\cic.dll	cic 1.0 Type Lib...
LM Auto Effect Behaivor	C:\WINDOWS\system32\lmrt.dll	

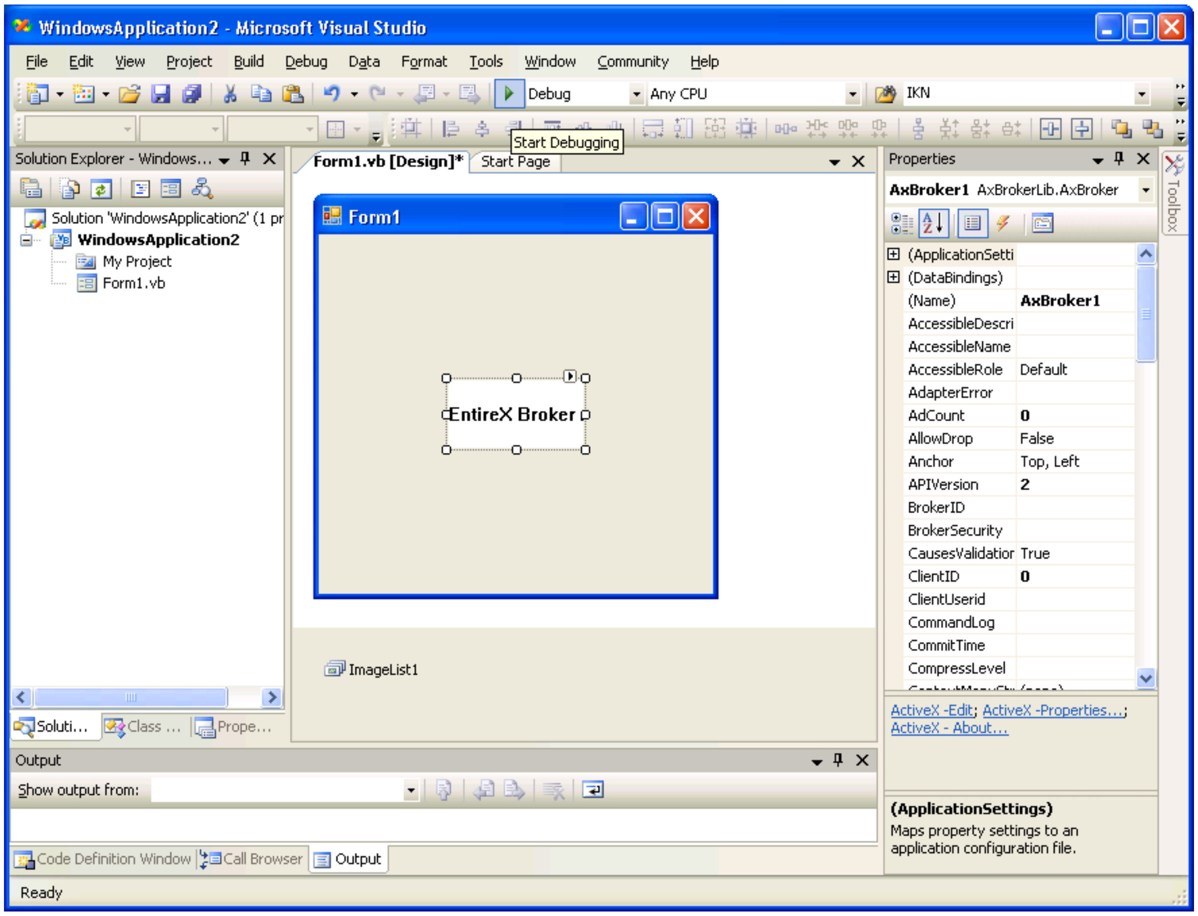
ActionBvr Class

Language: Language Neutral

Version: 1.0.0115



EntireX Broker ActiveX Control is now known to Visual Studio. It can be copied and pasted into the new form for later use.


WindowsApplication2 - Microsoft Visual Studio
File Edit View Project Build Debug Data Format Tools Window Community Help
Debug
Any CPU
IKN
Start Debugging
Solution Explorer - Windows...
Solution 'WindowsApplication2' (1 pr
WindowsApplication2
My Project
Form1.vb
Soluti...
Class ...
Prope...
Form1.vb [Design]*
Start Page
Form1
EntireX Broker
ImageList1
Properties
Toolbox
ActiveX -Edit; ActiveX -Properties...; ActiveX - About...
Output
Show output from:
Code Definition Window
Call Browser
Output
Ready

AxBroker1 AxBrokerLib.AxBroker

⊞ (ApplicationSetti	
⊞ (DataBindings)	
(Name)	**AxBroker1**
AccessibleDescri	
AccessibleName	
AccessibleRole	Default
AdapterError	
AdCount	**0**
AllowDrop	False
Anchor	Top, Left
APIVersion	**2**
BrokerID	
BrokerSecurity	
CausesValidatior	True
ClientID	**0**
ClientUserid	
CommandLog	
CommitTime	
CompressLevel	

(ApplicationSettings)
Maps property settings to an application configuration file.

Using the Property Pages

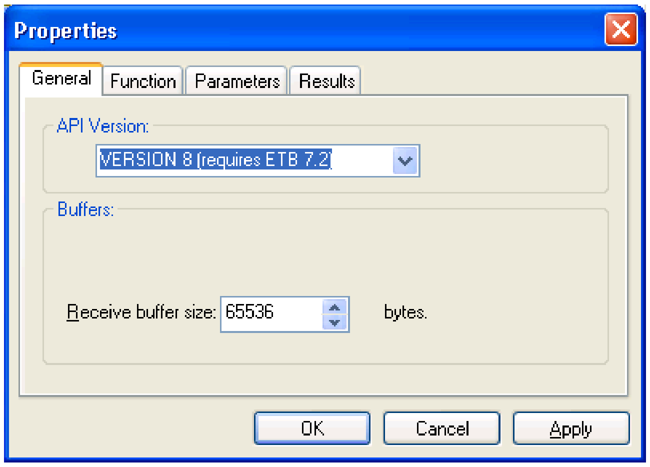
If you do not use Transaction Object Repository (TOR) files, you can also supply the properties using the property sheet of Broker ActiveX Control. (If you use Broker ActiveX Control as an automation server, the property pages are not available.)

The property sheet contains the following:

- General Page
- Function Page
- Parameters Page
- Results Page

General Page

With this page you can specify the API version and the size of the receive buffer.


Properties

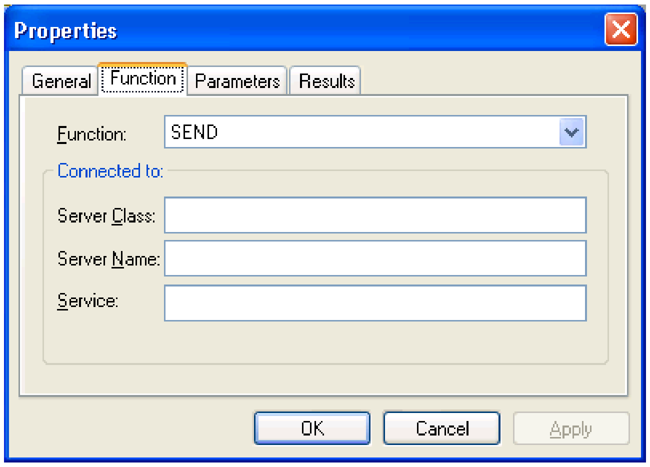
API Version: VERSION 8 (requires ETB 7.2)

Buffers:

Receive buffer size: 65536 bytes.

Function Page

With this page you can specify the function to be called and Service, Server Class and Server Name.



Function: SEND

Connected to:

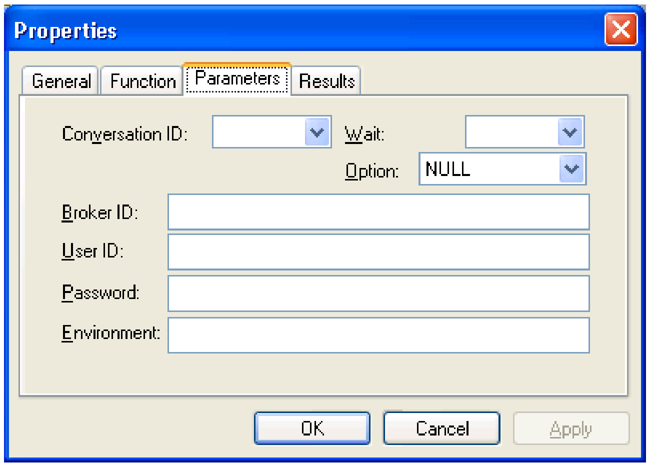
Server Class:

Server Name:

Service:

Parameters Page

With this page you can specify the Conversation ID, Broker ID, User ID, Password, Environment, Wait time, and Option.



Conversation ID:

Wait:

Option: NULL

Broker ID:

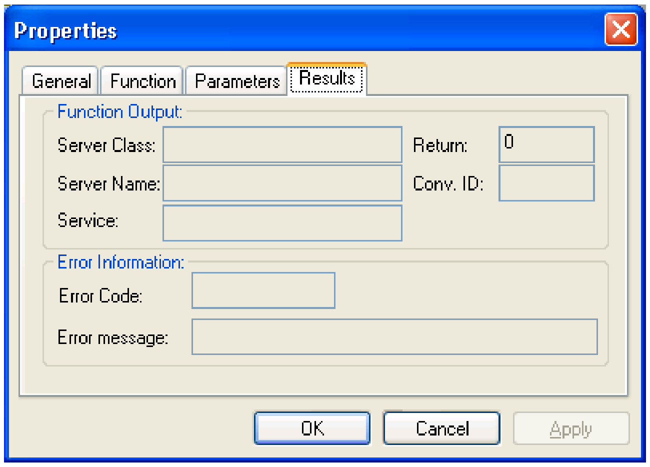
User ID:

Password:

Environment:

Results Page

This page displays the results of the Broker function.



Function Output:

Server Class:

Server Name:

Service:

Return: 0

Conv. ID:

Error Information:

Error Code:

Error message:

4 Broker ActiveX Control with Visual Basic

- Step 1: Instantiate EntireX Broker ActiveX Control 18
- Step 2: Instantiate the Transaction Object 20
- Step 3: Call Methods 20
- Step 4: Access the Returned Data 21
- Step 5: Cleanup Resources 24
- Step 6: Error Handling in Transaction Object Methods 24
- Examples: Writing an ACI Client and Server with Broker ActiveX Control 24

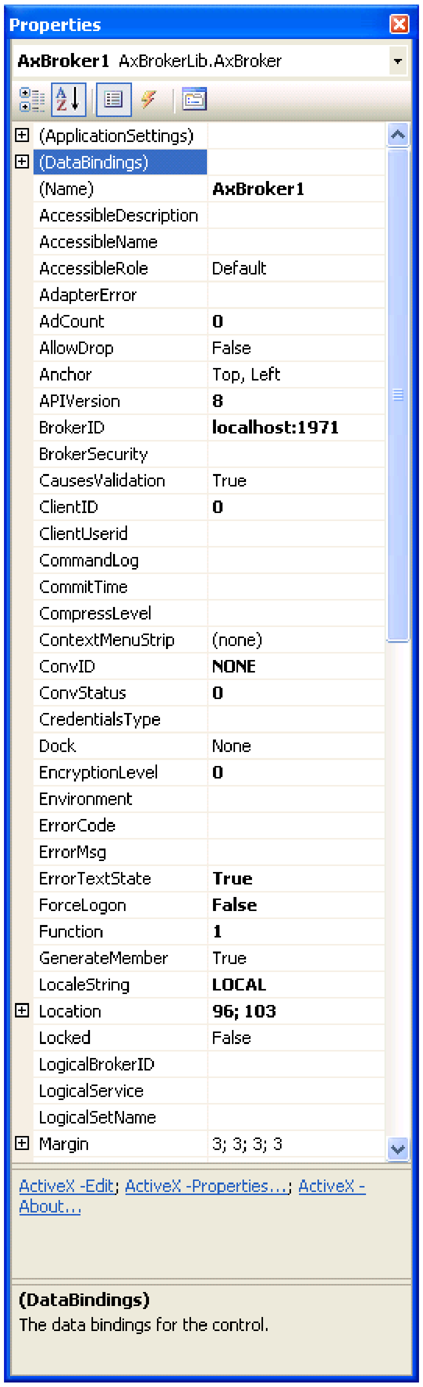
Visual Basic is used here as an example of a development environment in which applications using Broker ActiveX Control can work. Broker ActiveX Control can be used by any programming language or programming environment that can act as a container for ActiveX controls.

Note: If you edit a Visual Basic application that uses Broker ActiveX Control and save these changes with the new version of Broker ActiveX Control, you will not be able to use this application with Broker ActiveX Control version 1.2.1.

Step 1: Instantiate EntireX Broker ActiveX Control

To use Broker ActiveX Control as a control

1. From the **Project, Components, Controls** menu choose **EntireX Broker ActiveX Control**.
2. Drop it into your dialog.



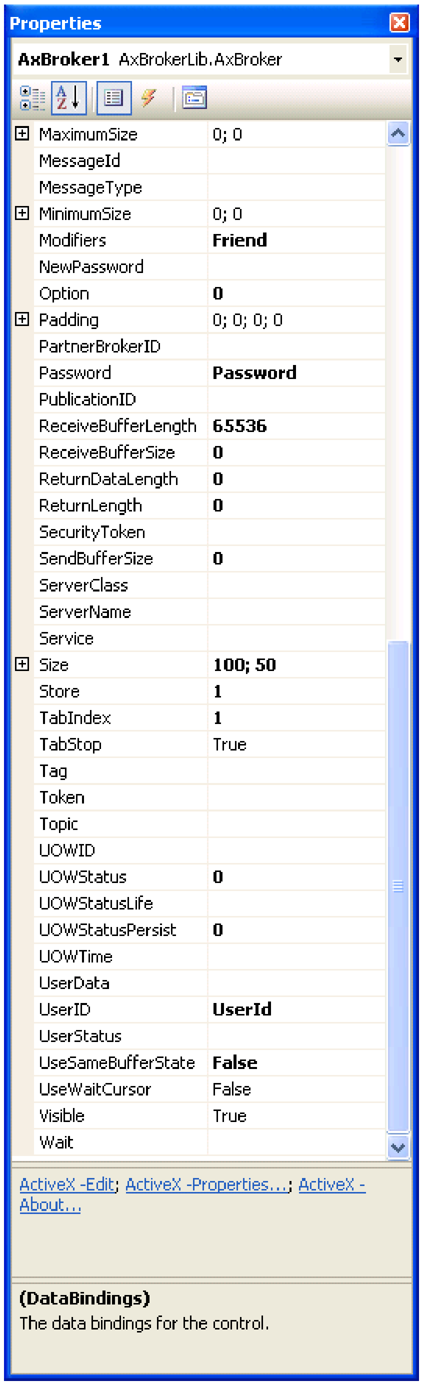
Properties

AxBroker1 AxBrokerLib.AxBroker

⊞ (ApplicationSettings)	
⊞ (DataBindings)	
(Name)	AxBroker1
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
AdapterError	
AdCount	0
AllowDrop	False
Anchor	Top, Left
APIVersion	8
BrokerID	localhost:1971
BrokerSecurity	
CausesValidation	True
ClientID	0
ClientUserid	
CommandLog	
CommitTime	
CompressLevel	
ContextMenuStrip	(none)
ConvID	NONE
ConvStatus	0
CredentialsType	
Dock	None
EncryptionLevel	0
Environment	
ErrorCode	
ErrorMsg	
ErrorTextState	True
ForceLogon	False
Function	1
GenerateMember	True
LocaleString	LOCAL
⊞ Location	96; 103
Locked	False
LogicalBrokerID	
LogicalService	
LogicalSetName	
⊞ Margin	3; 3; 3; 3

ActiveX -Edit; ActiveX -Properties...; ActiveX - About...

(DataBindings)
The data bindings for the control.



Properties

AxBroker1 AxBrokerLib.AxBroker

⊞ MaximumSize	0; 0
MessageId	
MessageType	
⊞ MinimumSize	0; 0
Modifiers	Friend
NewPassword	
Option	0
⊞ Padding	0; 0; 0; 0
PartnerBrokerID	
Password	Password
PublicationID	
ReceiveBufferLength	65536
ReceiveBufferSize	0
ReturnDataLength	0
ReturnLength	0
SecurityToken	
SendBufferSize	0
ServerClass	
ServerName	
Service	
⊞ Size	100; 50
Store	1
TabIndex	1
TabStop	True
Tag	
Token	
Topic	
UOWID	
UOWStatus	0
UOWStatusLife	
UOWStatusPersist	0
UOWTime	
UserData	
UserID	UserId
UserStatus	
UseSameBufferState	False
UseWaitCursor	False
Visible	True
Wait	

ActiveX -Edit; ActiveX -Properties...; ActiveX - About...

(DataBindings)
The data bindings for the control.

In this example, Name is set to "BOX" in the **Properties** dialog:

Using Broker ActiveX Control as an Automation Server

If you want to see the interface description of Broker ActiveX Control in the object browser or use the early bind feature:

From the **Project > References** menu, choose **Browse** and then select Broker ActiveX Control in *<drive>:\SoftwareAG\EntireX\bin\ebx.dll*.

To use Broker ActiveX Control as an automation server, you can define the following in your code:

```
Dim BOX as Object
```

or

```
Dim BOX as Broker
Set BOX=CreateObject("EntireX.Broker.ACI")
```

If you use Broker ActiveX Control as an automation server, you will not be able to:

- call the methods `DefineTOMethods` and `AboutBox`
- use the property pages.

Step 2: Instantiate the Transaction Object

If a Transaction Object Repository (TOR) file is used, it is not necessary to set the other properties. If you want to use a transaction object, instantiate the transaction object with the command:

```
Dim TransObject As Object
Set TransObject = BOX.CreateTransObject("c:\\path\\to\\trans\\object\\object.tor")
```

BOX is the name set previously.

See the ***Methods*** for list of methods available for supporting transaction objects.

Step 3: Call Methods

Once a transaction object has been instantiated, the methods defined in that transaction object can be called. If the transaction object method being called has one or more return values, transaction object methods *always* return these values wrapped in a return object.

```
Dim ReturnObject As Object
Set ReturnObject = TransObject.MyMethod("Param1", 50, "Param3")
```

A return object is always used, as TO methods usually return multiple scalar data items, or arrays, structures or records. These in fact define the possible return values in a return object. They will be either scalars:

- 2-byte INT

- 4-byte INT
- etc., basically all scalar types handled through the automation VARIANT structure

or objects:

- structure objects
- collection objects
- arrays
- records

Alias custom types are mapped internally to the data type they alias, either scalars or objects.

Step 4: Access the Returned Data

You then access the returned data by interpreting the return object. The code required depends on whether you are accessing scalars, structures, or arrays and records.

Note: Care must be taken to avoid recursive complex type definitions. For example, a structure should not be defined that contains an instance of itself, or less directly, an array of structures should not be defined that contains an instance of the same array type. These and other permutations of recursive definitions cannot be resolved, and thus cannot be used.

Scalars

Scalars can be accessed through the return object with code like this:

```
Dim Str As String
Dim Int As Integer
Str = ReturnObject.MyString
     Int = ReturnObject.MyInt
```

Structures

Structures can be accessed from the return object like this:

```
Dim Struct As Object
Dim Str As String
Set Struct = ReturnObject.MyStruct
     Str = Struct.MyString
```

Arrays and Records Exposed as Collections

Arrays and records are exposed by Broker ActiveX Control as automation collections when the method `CreateTransObject` is used. As collections, they support the `Count` property, as well as the `Item` property that acts as the default value when subscripting is performed without the `Item` name. Thus, an array in the return object can be accessed like this:

```
Dim Array_Value As Object
Dim I As Integer
Dim MyInt As Integer
Set Array_Value = ReturnObject.MyArray
For I = 0 To Array_Value.Count - 1
  MyInt = Array_Value(I)
Next I
```

The elements of a record can be accessed with the following method:

```
Dim Array_Value,Struct As Object
Dim I As Integer
Set Array_Value = ReturnObject.MyArray
For I = 0 To Array_Value.Count - 1
  Set Struct = Array_Value(I)
  Str = Struct.Str
Next
```

or also:

```
Dim Array_Value,Struct As Object
Dim I As Integer
Set Array_Value = ReturnObject.MyArray
For Each Struct in Array_Value
  Str = Struct.str
     Next
```

Arrays and Records Exposed as Safe Arrays

Arrays and Records are exposed as safe arrays when the method `CreateTransObjectSA(`*`torfilename`*`)` is used. Instead of the `Count` property, the `LBound` and `UBound` functions are supported.

An array in the return object can be accessed like this:

```
Dim Array_Value as Variant
Dim I as Integer
Dim Str as String
Array_Value = ReturnObject.MyArray
For I = LBound(Array_Value) To UBound(Array_Value)
  Str = Array_Value[I]
Next
```

The elements on a record can be accessed with the following method:

```
Dim Array_Value as Variant
Dim Struct as Variant
Dim I as Integer
Dim Str as String
Array_Value = ReturnObject.MyArray
For I = LBound(Array_Value) To UBound(Array_Value)
  Set Struct = Array_Value[I]
  Str = Struct.Str
Next
```

Another possible `For` statement:

```
For Each Struct in Array_Value
  Str = Struct.Str
Next
```

There are no limitations to the number of complex types or their relationship to each object in a transaction object. Arrays can exist within structures, and conversely, structures and arrays can exist within records, etc. Thus, multidimensional arrays can easily be simulated if the given Broker service that the method maps to provides data in such a format.

Step 5: Cleanup Resources

When objects in your automation code are no longer used, be sure to call:

```
Set ObjectName = Nothing
```

This decrements the reference count of the object, thus allowing cleanup of object resources. While the above information pertains specifically to Visual Basic, the concepts are also relevant to other automation controllers, such as Delphi and FoxPro.

Step 6: Error Handling in Transaction Object Methods

TO methods do not return an error flag; they raise a standard ActiveX exception instead. In Visual Basic, this exception can be caught with an 'On error' clause. The most likely reason for the failure of a TO method is that the Broker call that was issued returned an error. In Visual Basic, use the standard Err object to retrieve the error number and message (Err.Number and Err.Description).

If the error is a Broker error, Err.Description shows a generic error message "Automation Error". For a detailed error description use the `ErrorCode` and `ErrorMsg` properties.

Examples: Writing an ACI Client and Server with Broker ActiveX Control

- Writing an ACI Client with Broker ActiveX Control
- Writing an ACI Server with Broker ActiveX Control

Writing an ACI Client with Broker ActiveX Control

```
On Error Resume Next
Dim ebx As Object
Dim senddata As String
Dim loopcount As Integer
 loopcount = 0
' simple data to send
senddata = "Hello"
 Set ebx = CreateObject("EntireX.Broker.ACI")
ebx.BrokerID = "localhost"
ebx.ServerClass = "ACLASS"
ebx.ServerName = "ASERVER"
ebx.Service = "ASERVICE"
ebx.UserId = "EBX-USER"
ebx.function = 9 ' Logon
ebx.InvokeBrokerFunction
If ebx.ErrorCode <> 0 Then
MsgBox ebx.ErrorMessage
Exit Sub
End If
Do
ebx.function = 1 ' Send
ebx.ConvID = "NONE"
' SetSendData data, length of data
ebx.SetSendData senddata, Len(senddata)
ebx.wait = "10s" ' wait 10 seconds for a response from server
ebx.InvokeBrokerFunction
If ebx.ErrorCode <> 0 Then
MsgBox ebx.ErrorMsg
Else
MsgBox "Received " + Str(ebx.ReturnDataLength) + " bytes (" + ebx.GetReceiveData + ")"
End If
loopcount = loopcount + 1
If loopcount = 2 Then
senddata = " shutdown"
End If
Loop Until loopcount > 2
ebx.function = 10 ' Logoff
ebx.InvokeBrokerFunction
If ebx.ErrorCode <> 0 Then
MsgBox ebx.ErrorMessage
     End If
```

Writing an ACI Server with Broker ActiveX Control

```
On Error Resume Next
Dim ebx As Object
Dim senddata As String
Dim receivedata As String
' simple data to send
senddata = "Hello"
Set ebx = CreateObject("EntireX.Broker.ACI")
ebx.BrokerID = "localhost"
ebx.ServerClass = "ACLASS"
ebx.ServerName = "ASERVER"
ebx.Service = "ASERVICE"
ebx.UserId = "EBX-USER"
ebx.function = 9 ' Logon
ebx.InvokeBrokerFunction
If ebx.ErrorCode <> 0 Then
MsgBox ebx.ErrorMessage
Exit Sub
End If
ebx.function = 6 ' Register
ebx.InvokeBrokerFunction
```

```
If ebx.ErrorCode <> 0 Then
MsgBox ebx.ErrorMessage
End If
Do
ebx.function = 2 ' Receive
ebx.wait = "yes" ' wait until data is received
ebx.ConvID = "NEW"
ebx.SetReceiveBufferLength = 1024 ' we are now able to receive messages up to 1024 ↩
bytes
ebx.InvokeBrokerFunction
If ebx.ErrorCode <> 0 Then
MsgBox ebx.ErrorMsg
Else
' save received data
receivedata = ebx.GetReceiveData
' send response
ebx.function = 1 ' Send
' SetSendData data, length of data
ebx.SetSendData senddata, Len(senddata)
ebx.wait = "no" ' don't wait for a response
ebx.InvokeBrokerFunction
If ebx.ErrorCode <> 0 Then
MsgBox ebx.ErrorMsg
Else
MsgBox "Received data: " + receivedata
End If
End If
' loop until the received data has the string "shutdown" from the position 20
receivedata = Mid(receivedata, 20, 8)
Loop Until receivedata = "shutdown"
ebx.function = 7 ' DeRegister
ebx.InvokeBrokerFunction
If ebx.ErrorCode <> 0 Then
MsgBox ebx.ErrorMessage
End If
ebx.function = 10 ' Logoff
ebx.InvokeBrokerFunction
If ebx.ErrorCode <> 0 Then
MsgBox ebx.ErrorMessage
     End If
```

5 Using Broker ActiveX Control with Active Server Pages

- Prerequisites 28
- Designing a Web Page with ASP and Broker ActiveX Control 28
- Using Broker ActiveX Control in Multiple Pages 30

Microsoft's Active Server Page (ASP) is an HTML page that includes one or more scripts and reusable ActiveX server components to create dynamic Web pages. The scripts and ActiveX components are processed on a Microsoft Web server before the page is sent to the user.

Prerequisites

Installation prerequisites for all EntireX components are described centrally. See *Prerequisites* in the Release Notes.

To use Broker ActiveX Control with ASP, you must have a running Web server.

Designing a Web Page with ASP and Broker ActiveX Control

Creating an Instance of the ActiveX Control and the Transaction Object

```
<%
Set EBX    = server.Createobject("EntireX.Broker.ACI")
Set torobj = EBX.CreateTransObject("calc.tor")
```

or

```
Set torobj = EBX.CreateTransObjectSA("calc.tor") (if returnvalue contains array)
     %>
```

Calling a TOR Method

```
Set retobj = torobj.calc(op,op1,op2)
```

Accessing the Data

Scalars

```
     <% string = retobj.result %>
```

Structures

```
     <% string = retobj.result.str %>
```

Arrays

You can have access to array elements:

```
<%string = retobj.retarr(0) %>
```

or

```
<%
return = retobj.retarr
string = return(0)
%>
```

or

```
<%
For Each element in retobj.retarr
  string = element
Next
      %>
```

Records

You can have access to record elements:

```
<%string = retobj.retrec(0).str %>
```

or

```
<%
Set return = retobj.retrec(3)
Response.Write return.str
%>
```

or

```
<%
For Each struct in retobj.retrec
  string = struct.str
Next
%>
```

or

```
<%
Array_Value = retobj.retrec
For I = LBound(Array_Value) To UBound(Array_Value)
  string = Array_Value(I).str
Next
      %>
```

Using Broker ActiveX Control in Multiple Pages

Objects created by `Server.CreateObject` or `CreateTransObject` have page scope. They will be destroyed automatically when the current ASP page is finished.

To create an object with session or application scope, you can either use the `<OBJECT>` tag and set the `SCOPE` parameter to SESSION or APPLICATION, or store the object in a session or application variable.

For example, an object stored in a session variable, as shown in the following script, is destroyed when the Session object is destroyed. That is, when the session times out, or the `Abandon` method is called.

```
<% Set Session("torobj") = EBX.CreateTransObject("calc.tor")%>
```

You can destroy the object by setting the variable to "Nothing" or setting the variable to a new value.

```
<% Session("torobj") = Nothing %>
```

6 Using Broker ActiveX Control with .NET

- Using Broker ActiveX Control with Visual Studio .NET 32
- A Small Visual Basic .NET Example 32

Using Broker ActiveX Control with Visual Studio .NET

To use Broker ActiveX Control with Visual Studio .NET

1 Add Broker ActiveX Control to the Project references.

2 Add a Broker Control variable `BrokerLib.BrokerClass()`.

While you are using Broker ActiveX Control, the properties and methods of the object are listed in the member list.

Note: To use custom data types you have to access the items through a temporary object. See *Defining Custom Data Types*.

A Small Visual Basic .NET Example

```
' create new ActiveX Control
Dim broker As New BrokerLib.BrokerClass()
Dim TransactionObject As Object
Dim SomeObject As Object
Dim CTObject As Object
' load tor object
TransactionObject = broker.CreateTransObject("Broker.tor")
' call a method from the tor object
SomeObject = TransactionObject.GetData("Person1")
'
reference a temporary object to the Customer Data type
CTObject = SomeObject.CustData
' access to the items of the Customer Data
Console.WriteLine("Name   :" & CTObject.Name)
    Console.WriteLine("Address :" & CTObject.Address)
```

7 Transaction Objects in Broker ActiveX Control

- Advantages of Transaction Objects 34
- Calling the Transaction Object Editor 34
- Managing TOR Files 36
- Defining Methods 39
- Specifying Connection Information 44
- Defining Custom Data Types 46

Transaction Object (TOs) in Broker ActiveX Control are selections of logical methods that are stored in a transaction object repository (TOR). These logical methods contain all the connection and interface details necessary to communicate with EntireX Broker.

Advantages of Transaction Objects

The advantages of using transaction objects are:

- Services are defined once, in one place, and distributed as needed. They can then be used by anyone from many different applications to access back-end applications.
- Transaction objects can encapsulate all connection and conversational information from the developer, which simplifies the implementation and administration of distributed applications.
- The *send* buffer of a message is broken down into parameters, and the *receive* buffer is mapped to the return object. This means you do not have to worry about offsets, data types, repeating fields (arrays), or structures.

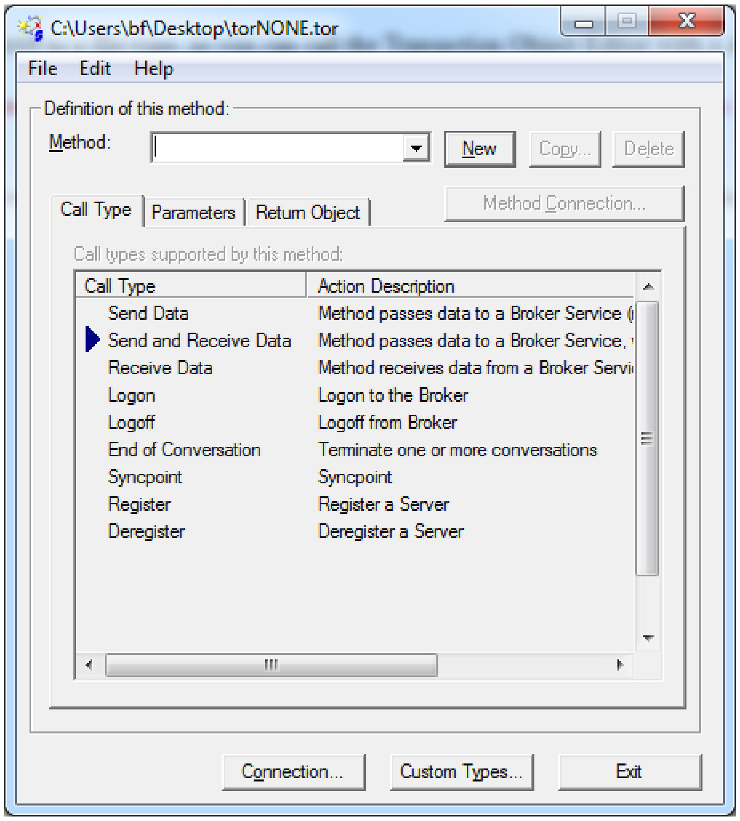
Calling the Transaction Object Editor

The Transaction Object Editor is a tool within Broker ActiveX Control with which you can define and maintain transaction objects. It is invoked by calling the method `DefineTOMethods` from a form that includes an ActiveX control.

The Transaction Object Editor can be called directly using the `TORedit` executable. The extension ".tor" is registered as a file type, so you can call the Transaction Object Editor with a double click from the Windows Explorer.

Notes:

1. After installation you will find the Transaction Object Editor in directory *<inst_root>\EntireX\bin\x86* (32-bit).
2. Before you start the TOR Editor for the first time, you need to register the required DLL *ebx.dll* to your Windows system manually. Simply open a Windows Command prompt in folder *<inst_root>\EntireX\bin\x86* and run the command `regsvr32 ebx.dll`. If you later want to use a TOR Editor from a different installation directory, register the corresponding *ebx.dll* as above.



C:\Users\bf\Desktop\torNONE.tor

Call Type	Action Description
Send Data	Method passes data to a Broker Service (
Send and Receive Data	Method passes data to a Broker Service,
Receive Data	Method receives data from a Broker Servi
Logon	Logon to the Broker
Logoff	Logoff from Broker
End of Conversation	Terminate one or more conversations
Syncpoint	Syncpoint
Register	Register a Server
Deregister	Deregister a Server

When a transaction object is loaded, the corresponding file name will be displayed in the title bar. If loading or saving fails, an error message will be displayed in the title bar.

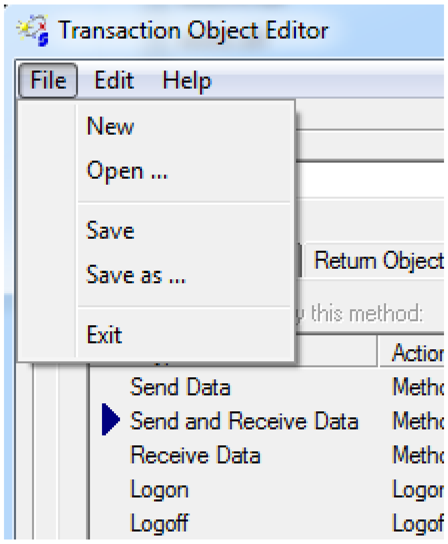
Managing TOR Files

The following functions are available for managing TOR files.

- File Menu
- Edit Menu

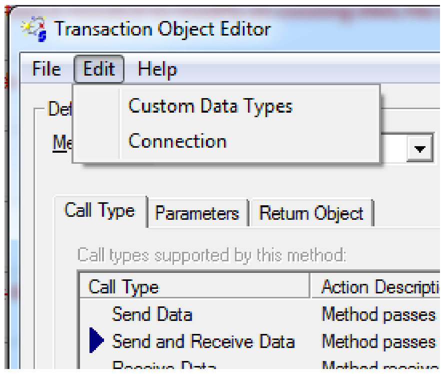
- Help Menu

File Menu



Menu Item	Description
New	Resets the TOR Editor.
Open	Loads an existing TOR file. A standard **Open File** dialog will be displayed. This function is needed to modify an existing TOR file.
Save	Saves a TOR file.
Save as	Saves a new or modified TOR file. A standard **Save File** dialog will be displayed.
Exit	Closes the TOR Editor.

Edit Menu



Menu Item	Description
Custom Types	Calls the **Custom Data Types** dialog. See *Defining Custom Data Types*.
Connection	Calls **Broker Connection Information dialog**. See the *Specifying Connection Information*

Help Menu

Menu Item	Description
About	Displays the **About** box.

Defining Methods

The following buttons are available in the transaction method definition model:

- The **New** button causes the method name within the dialog box to be added to the store.
- The **Copy** button copies the currently selected method to a new method.
- The **Delete** button removes the selected method from the store.

Methods are logically grouped in a transaction object. Each method specified in the transaction object relates directly to a specific Broker service. To define a new method, therefore, you need to know which services are available. Each method requires the following information:

- Connection
- Call Type
- Parameters
- Return Object

Connection

Connection information is specified using the **Broker Connection Information** dialog. Each TOR file has default connection information, and each method has its individual connection information. If a parameter is not defined in the connection information of a method, the default is taken. For a description of the parameters, see *Specifying Connection Information*.

Call Type

The **Call Type** tab represents the call types that can be used for this method.

Call Type	Description
Send Data	Used to define a method that accepts parameters but does not return data from the service. This could be used to notify a back-end application of some event without waiting for a response.
Send and Receive Data	Used to define a method that accepts parameters and returns data from that service.
Receive Data	Can be used to get information from a back-end application that requires no input, for example MOTD (message of the day) information. It is also used to wait for incoming requests if you are using Broker ActiveX Control to write Broker Server applications.
Logon	Logon to EntireX Broker.
Logoff	Logoff from EntireX Broker.
End of Conversation	Used to end a conversation.
Syncpoint	Used to commit, backout, or cancel a unit of work, obtain the status of a unit of work, or delete the persistent status of a unit of work.

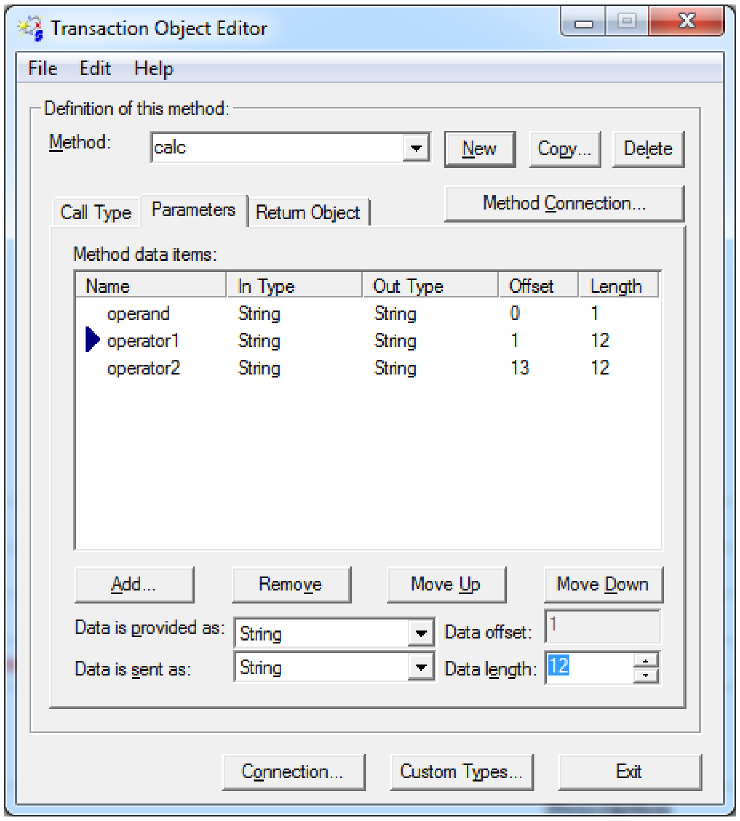
Call Type	Description
Register	Informs EntireX Broker that a service is available.
Deregister	Removes previously registered services from EntireX Broker's active list.

The **Call Type** tab is shown in the Transaction Object Editor screen above.

Parameters

The **Parameter** tab exposes a multiline box containing individual parameter variables.

These parameters are placed into the SEND-BUFFER of the EntireX Broker call. Each parameter has a data type (Integer, Real, String etc.) and a length.



Method: calc

Name	In Type	Out Type	Offset	Length
operand	String	String	0	1
operator1	String	String	1	12
operator2	String	String	13	12

Data is provided as: String
Data offset: 1
Data is sent as: String
Data length: 12

Defining a Parameter List

If data is sent, it is necessary to define a parameter list for this method. The `TO` method parameter list serves as a "map" between the types passed as parameters, and the data types and locations within the method's send buffer. Items within the `TO` method parameter list are ordered sequentially as they will be passed when the method is invoked.

List Control

A list control is used for defining, removing and ordering parameters of the current method. The list control supports in-place editing of items names, and works together with the item configuration controls positioned below. When a particular item is selected, it can be moved up and down the list sequentially. The order of the list defines the order in which parameters are passed when the method is invoked. Note that offsets are automatically generated for each list item, relative to the start of the list, and the items (and their sizes) that precede it.

The `Add` function adds the field after the selected position.

Data Conversion

Data conversion is also supported between a type provided by the client and the type expected by the Broker service. For parameters, the user can specify the data type that will be provided, and the type that will be sent to the Broker service. For return objects, the data received by the Broker service can be set to the data type retrieved by the user. The important data types are those sent to and received from a Broker service. Broker ActiveX Control automatically converts between the data type received from the Broker and a data type specified by the user (see the **Data is received as** and **Data is retrieved as** fields in the screen below).

Implemented Data Types

The scalar data types supported by the Broker ActiveX are a subset of the standard Automation VARIANT types and are listed below. In cases where the selected data type is of fixed length, the data length edit control is set to the appropriate length and grayed.

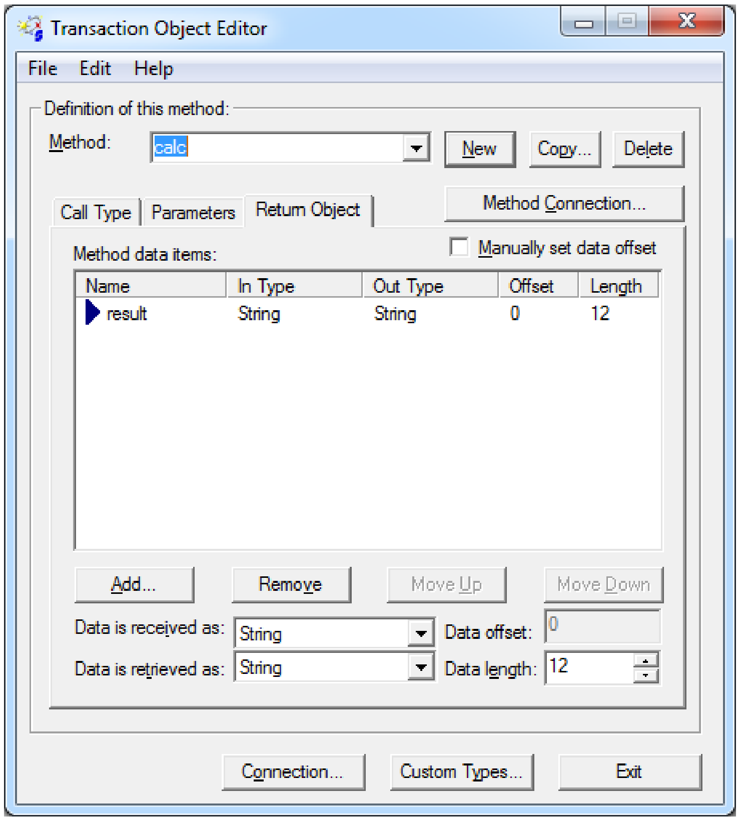
Transaction Object Method Data Types	Description
1-byte Integer	1-byte Integer used for signed and unsigned.
2-byte Integer	2-byte Integer used for signed and unsigned.
4-byte Integer	4-byte Integer used for signed and unsigned.
4-byte Real	4-byte Real compatible with "C" float.
8-byte Real	8-byte Real compatible with "C" double.
Bool	Boolean variable.
String	String of specified length.
Blob	Generic byte block.

Transaction Object Method Data Types	Description
Padding	Used to separate types in the buffer.

Return Object

If the transaction object method is invoked with call type 'Send and Receive' or 'Receive', a Return Object is created. This is a logical object that enables you to retrieve multiple scalar values or records by referencing its properties.

The **Return Object** tab exposes the individual properties that are mapped onto the RECEIVE-BUFFER of the Broker call. When the data is returned from the Broker service, Broker ActiveX Control uses the types and lengths of the defined properties to populate the values of the properties. You can now access the contents of the receive buffer as ActiveX properties of the method that is created by loading the transaction object.



Method: calc

Method data items:

☐ Manually set data offset

Name	In Type	Out Type	Offset	Length
result	String	String	0	12

Data is received as: String

Data is retrieved as: String

Data offset: 0

Data length: 12

As with the parameters, Broker ActiveX Control calculates the offset in the RECEIVE-BUFFER for each property. For information on list control, data conversion and implemented data types, see *Defining a Parameter List*.

Custom Data Types are used for non-scalar data types such as arrays and structures. They are also used to assign aliases to parameters for consistent naming purposes.

The **Manually set data offset** check box allows the transaction object designer to override automatic offset calculation and specify offsets manually. This feature is powerful, but also potentially dangerous, because no base type checking can be performed.

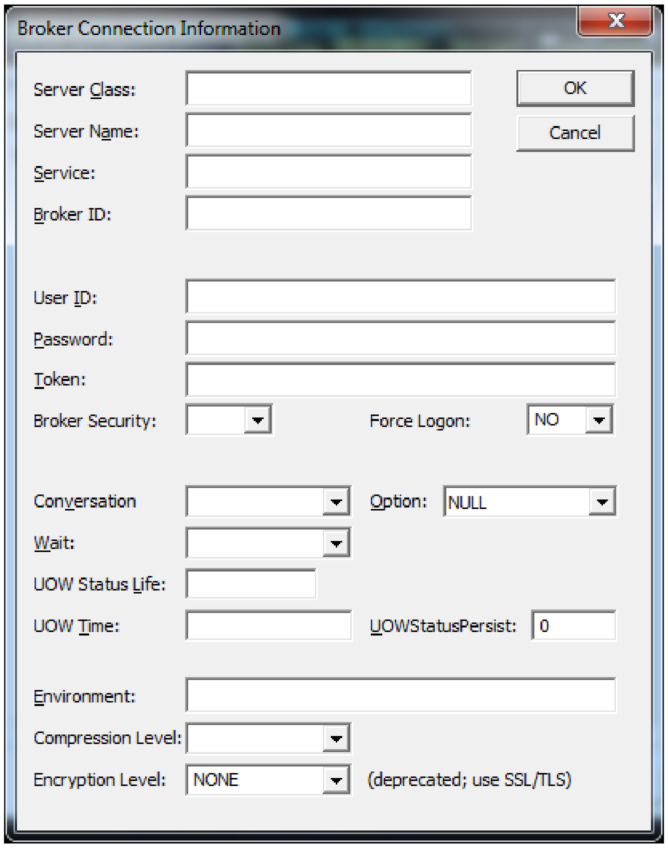
Specifying Connection Information

- Introduction
- Connection Information Parameters

Introduction

Connection information relates directly to the Broker service that you want to communicate with when using this method.

Transaction methods are defined using the Transaction Object Editor. Connection information is specified using the **Broker Connection Information** dialog. Each TOR file has default connection information, and each method has its individual connection information. If a parameter is not specified in the connection information of a method, the default is taken. The Broker parameters are part of this connection information (with the exception of `Function`, which depends on the Call Type).



Force Logon: NO

Option: NULL

UOWStatusPersist: 0

Encryption Level: NONE

The **Broker Connection Information** dialog box accepts all the parameters required for establishing the necessary Broker connection to execute the defined method/call type. Information in this dialog can be changed without affecting the application code. For example, if the BrokerID changed, you would change the connection information in the methods (services) affected and distribute the new transaction object file. The next time the application code loads the transaction object file and calls a method, the new connection information will be used.

Connection Information Parameters

Parameter	Description
ServerClass, ServerName, Service	These three parameters represent the unique "signature" of this method call.
BrokerID	The unique name of the Broker node that the services are attached to. See *Using the Broker ID in Applications* in the ACI Programming documentation and details on TCP/IP under *Transport Methods*.
Wait	The following values are set for this parameter, depending on the operation: Operation — Wait Value (in seconds) Send — 0 Send and Receive — 30 (*) Receive — 59 (*) (*) if no value is specified in the **Connection** info.

See ***Properties*** for a description of the other parameters.

Setting the Broker Call Parameters

Calling a method of a transaction object results in a Broker call. The parameters for the Broker call are taken either

- from the **Broker Connection Information** dialog, see above, or
- from the properties (see ***Properties***).

If a value is specified in the **Connection Information** dialog, this value is taken and overrides any value specified in the properties.

If no value is specified in the **Connection Information** dialog, the current setting of the properties is taken. Leaving these parameters blank in the **Connection Information** dialog enables you to change these parameters dynamically, and also enables Broker communication in conversational mode. See example below:

Visual Basic Example

This example shows a possible usage of dynamic parameter assignment:

```
Set TransObject=BOCX.CreateTransObject ("...calc.tor")
BOCX.UserID = "USER1"
BOCX.BrokerID = "ETB121"
      Set ReturnOb = TransObject.calc("+", "000000000001", "000000000002")
```

Defining Custom Data Types

The **Custom Data Types** dialog allows you to define new data types that will appear in the **Return Object** tag. With the **Apply** button you can embed a custom type within another custom type as long as this does not result in a recursive inclusion.

The following four classes of custom data types are supported:

- Custom Data Type 'Alias'
- Custom Data Type 'Array'
- Custom Data Type 'Record'
- Custom Data Type 'Structure'

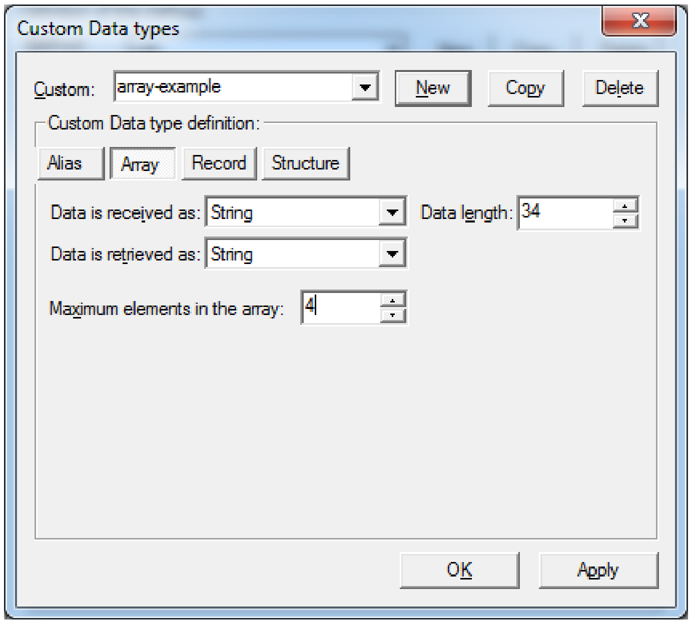
Any custom data type can be used in transaction objects return objects. Custom data types are not supported as method parameters.

Note: All custom data types can be used recursively. That is, any custom data type can be used as a member or base type for any other custom type. This allows for nested structures, as well as arrays within structures and records.

Custom Data Type 'Alias'

An *alias* is a custom data type that allows an administrator to specify an alias for any defined data type - custom or not. Aliasing also allows the definition of data types with specific in and out data types (type translation).

Custom Data Type 'Array'



Custom: array-example

Custom Data type definition:

Data is received as: String

Data length: 34

Data is retrieved as: String

Maximum elements in the array: 4

An *array* consists of multiple serial elements of the same data types. Arrays can be made up of either scalar or custom data types. The number of elements in an array must be specified.

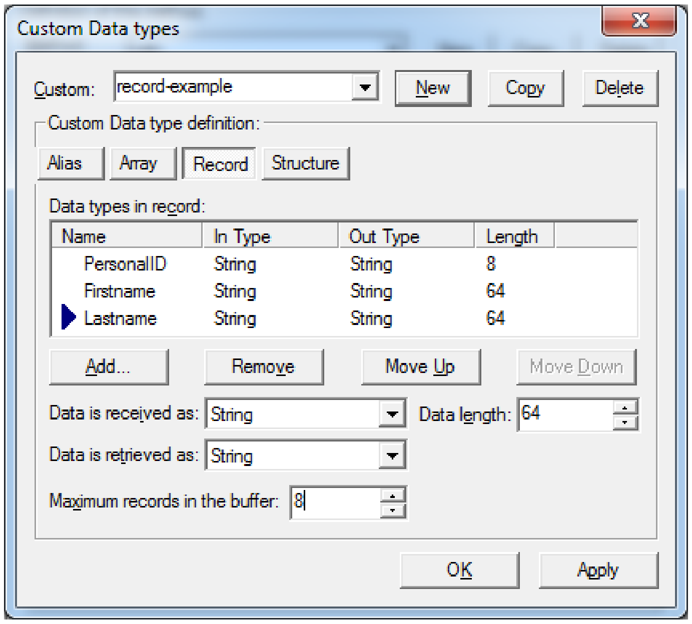
Array custom data types accept the same basic information as alias data types, with the addition of the number of elements in the array. Arrays allow elements of the specified base type to be accessed in a subscripted fashion.

Note: Multidimensional arrays and arrays of structures can be implemented by specifying a custom array or record data type as the base type of this array.

Custom Data Type 'Record'

A *record* is a repeating collection of data types - scalar or custom.

This custom data type allows you to define a collection of data types that can be accessed in a subscripted fashion. The order of defined types in the **Record** can be changed. Also, the number of records within the receive buffer can be specified if known.


Custom Data types

Custom: record-example

Custom Data type definition:

Data types in record:

Name	In Type	Out Type	Length
PersonalID	String	String	8
Firstname	String	String	64
Lastname	String	String	64

Data is received as: String

Data length: 64

Data is retrieved as: String

Maximum records in the buffer: 8

Custom Data Type 'Structure'

A *structure* is a named collection of data types.

The controls for this custom data type are identical to those of the data type 'record', with the exception of a repetitive count, which is not applicable.

8 Calling Broker ActiveX Control Remotely

- Setting up the Server Environment 50
- Setting up the Client Environment 55
- Testing the Connection 57

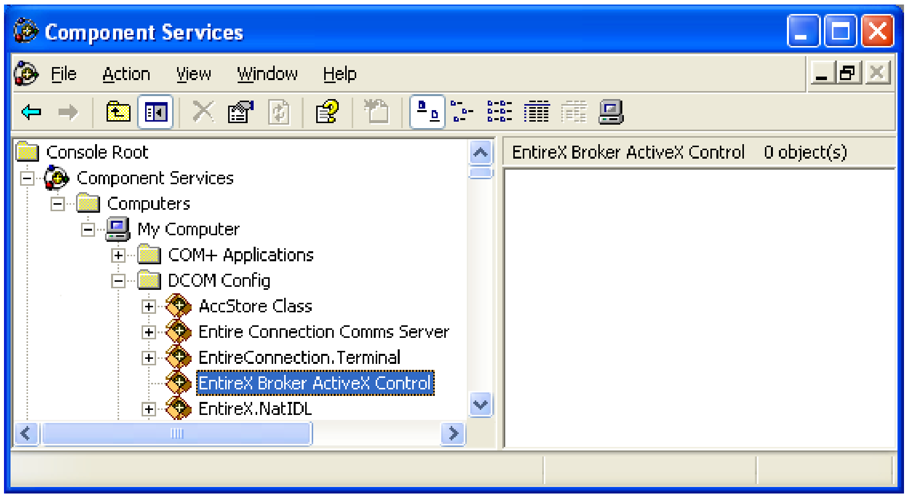
You can call Broker ActiveX Control remotely if you use it as an automation server. This means you can use the Broker component from a separate process - either on the same machine or on another machine in the network.

Setting up the Server Environment

To set up the server environment

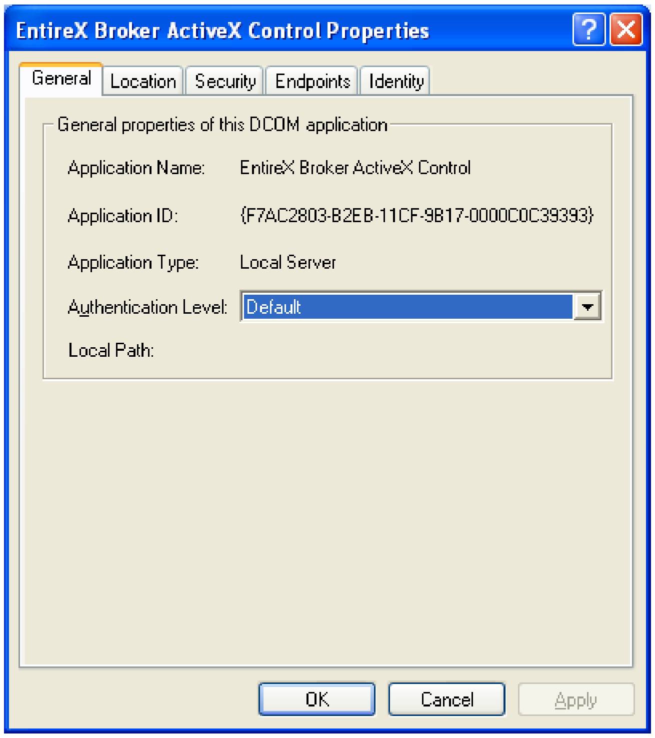
1 From the **Administrative Tools** in the **Control Panel**, open the **Component Services** on the server.

The following dialog box will be displayed:



Select **EntireX Broker ActiveX Control** in the **DCOM Config** list box and choose the properties from the context menu.

The following dialog box will be displayed:



General properties of this DCOM application

Application Name: EntireX Broker ActiveX Control

Application ID: {F7AC2803-B2EB-11CF-9B17-0000C0C39393}

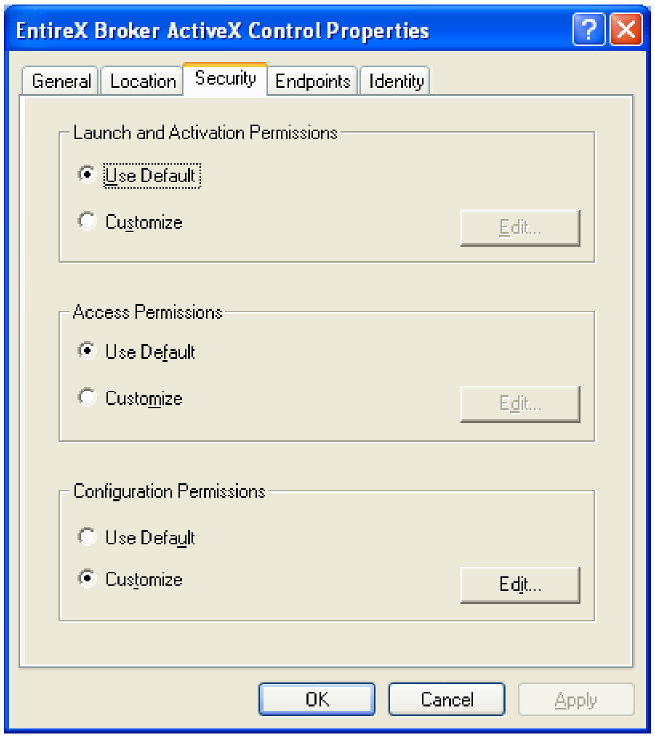
Application Type: Local Server

Authentication Level: Default

Local Path:

2 Click the **Security** tab.

Click the **Security** tab.



Launch and Activation Permissions

(●) Use Default

( ) Customize

Access Permissions

(●) Use Default

( ) Customize

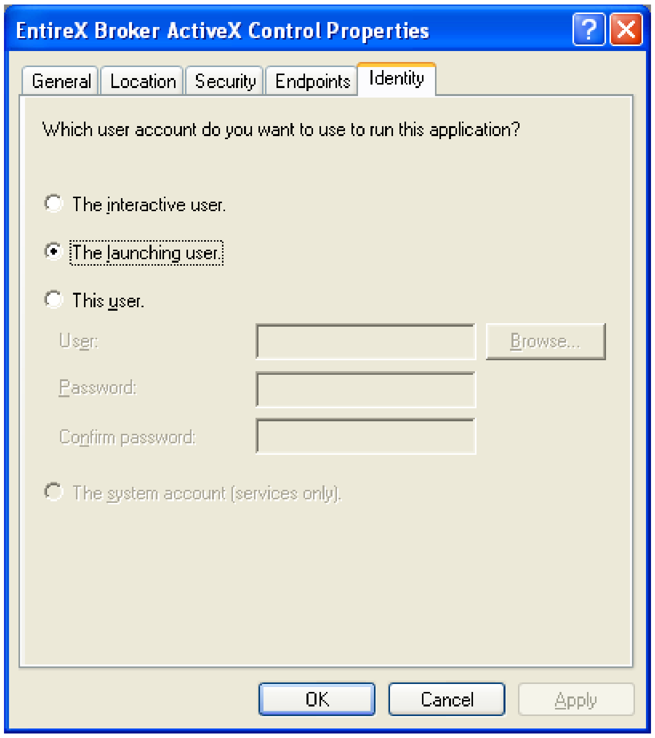
Configuration Permissions

( ) Use Default

(●) Customize

In the dialog box displayed above, keep the defaults for access, launch and configuration permissions.

3 Click the **Identity** tab.



Which user account do you want to use to run this application?

The interactive user.

[✓ The launching user.]

This user.

User:

Password:

Confirm password:

The system account (services only).

There are three options to define the user account to be used to run the application:

- **The interactive user**

  This implies that a user with permission to launch the application must be logged on to the server machine.

- **The launching user**

  This implies that an account must be created on the server machine with the same username/password as on the client machine. This account will then be used to launch the application.

- **This user**

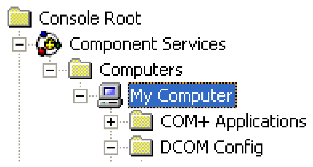
  A final option is to specify a user account to be used when launching the application.

In each case, the username/password of the client machine must also exist on the server machine.

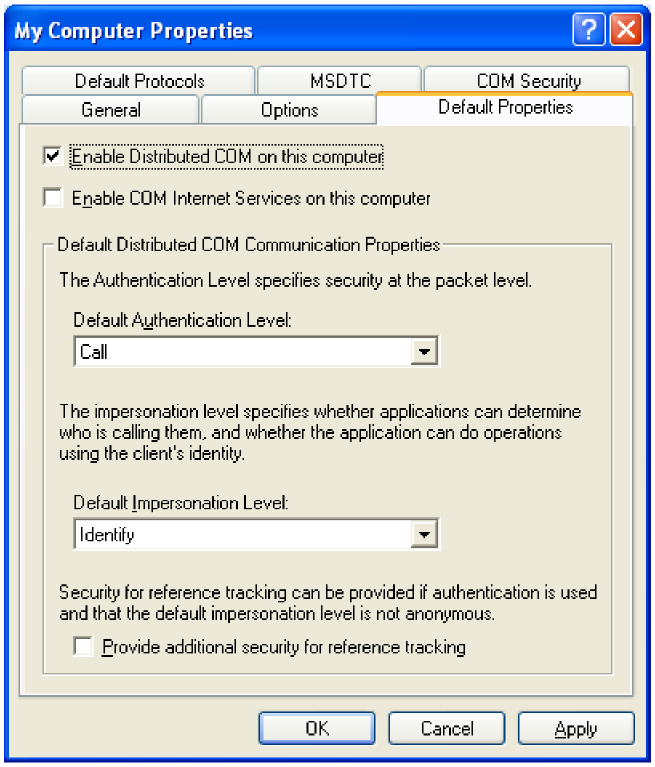
Select one of the options and choose **OK** to return to the **Component Services**.

4 Click on **My Computer** and choose the properties from the context menu.

Click on **My Computer** and choose the properties from the context menu.



The following dialog box will be displayed; click on the **Default Properties** tab.



[✓] Enable Distributed COM on this computer

[ ] Enable COM Internet Services on this computer

Default Distributed COM Communication Properties

The Authentication Level specifies security at the packet level.

Default Authentication Level: Call

The impersonation level specifies whether applications can determine who is calling them, and whether the application can do operations using the client's identity.

Default Impersonation Level: Identify

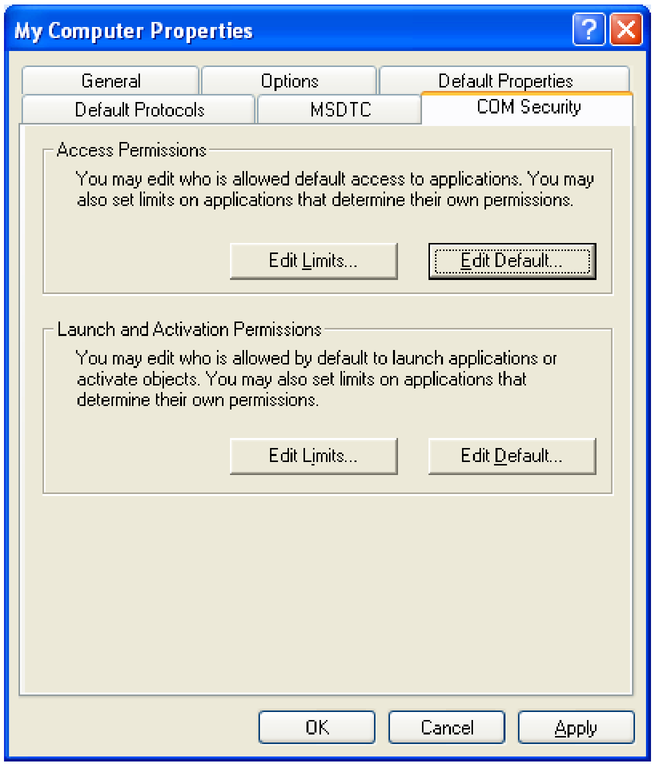
Security for reference tracking can be provided if authentication is used and that the default impersonation level is not anonymous.

[ ] Provide additional security for reference tracking

Choose the options as shown in the dialog box above.

5 Click on the COM Security tab.

Click on the **COM Security** tab.



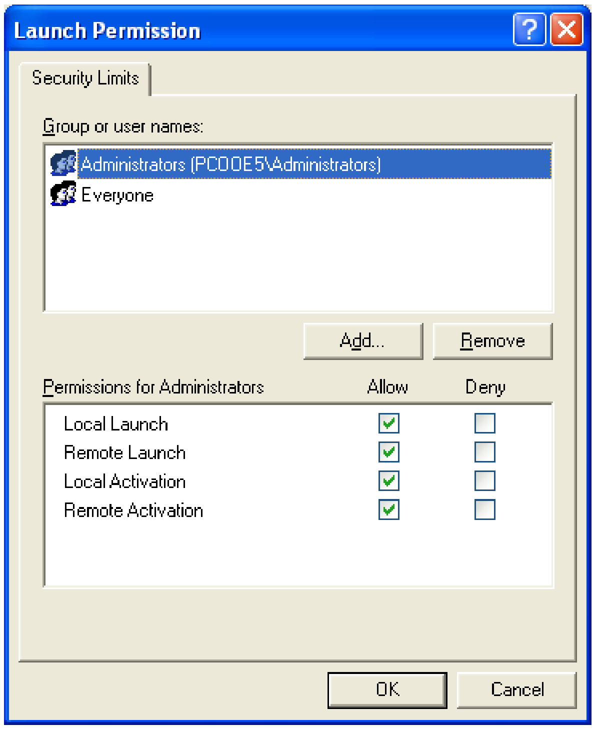
Access Permissions

You may edit who is allowed default access to applications. You may also set limits on applications that determine their own permissions.

Launch and Activation Permissions

You may edit who is allowed by default to launch applications or activate objects. You may also set limits on applications that determine their own permissions.

In the **Launch and Activation Permissions** area of the dialog box displayed above, choose **Edit Default**. The following dialog box will be displayed:



Group or user names:

Administrators (PC00E5\Administrators)
Everyone

Permissions for Administrators	Allow	Deny
Local Launch	✓	
Remote Launch	✓	
Local Activation	✓	
Remote Activation	✓	

Make sure that either the user corresponding to the client machine account, or a group to which the user belongs, has **Allow Launch** as **Type of Access**.

Choose **OK** in this screen and then **Apply**, and exit **Component Services** on the server.

Setting up the Client Environment

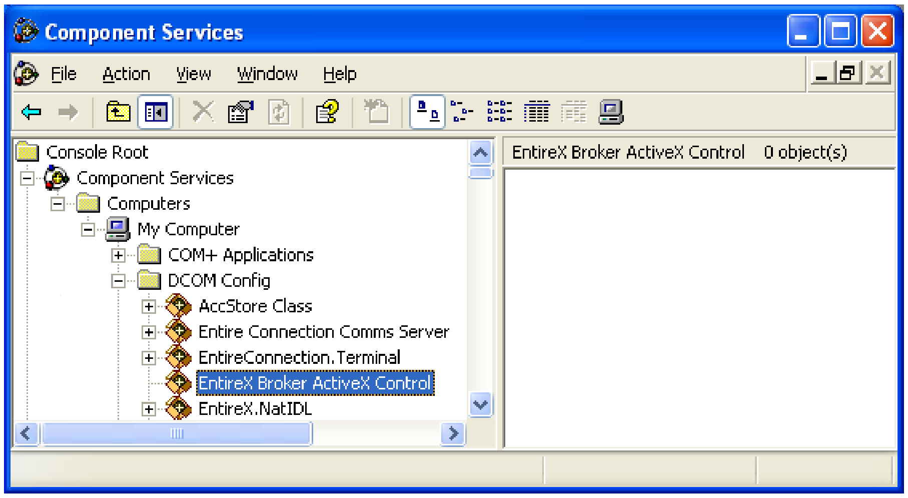
The *EbxProxy.dll* is installed by default on the server in directory *<drive>:\SoftwareAG\EntireX\bin*. Copy the file from the server machine to the client machine.

The DLL must then be registered with: `REGSVR32 <path>\EBXproxy.dll`.

To configure the client environment

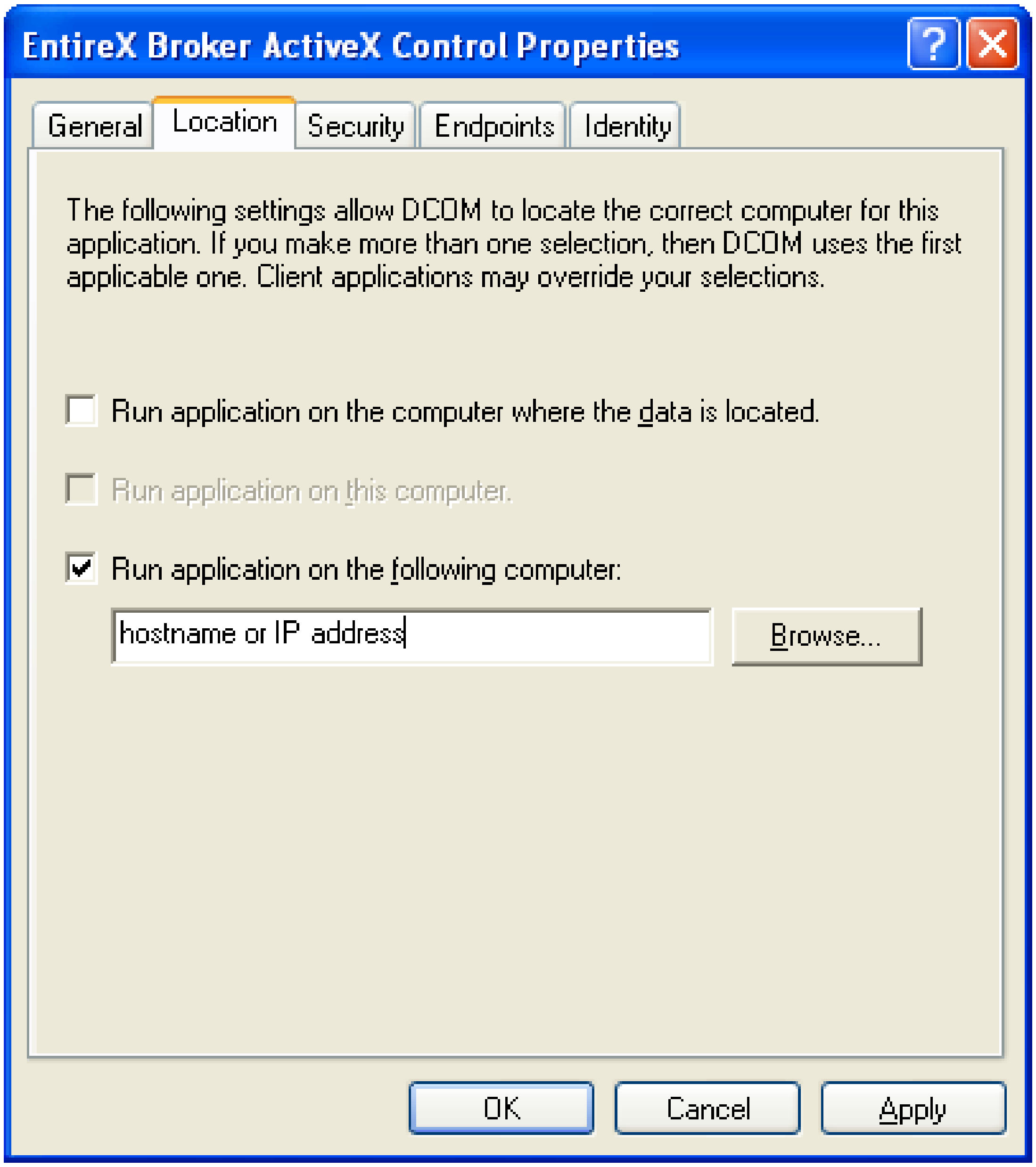
1 From the **Administrative Tools** in the Control Panel, open the **Component Services** on the client.

The following dialog box will be displayed:


Component Services
File Action View Window Help
Console Root
Component Services
Computers
My Computer
COM+ Applications
DCOM Config
AccStore Class
Entire Connection Comms Server
EntireConnection.Terminal
EntireX Broker ActiveX Control
EntireX.NatIDL
EntireX Broker ActiveX Control 0 object(s)

Select **EntireX Broker ActiveX Control** in the **DCOM Config** list box, choose the properties from the context menu and click the **Location** tab.

2 In the **Location** tab of the ActiveX Control **Properties** dialog box above, select the checkbox **Run application on the following computer:** and enter either the hostname or the IP address of the server machine.


EntireX Broker ActiveX Control Properties

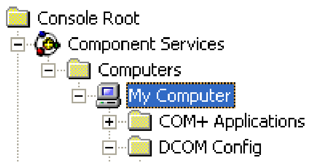
The following settings allow DCOM to locate the correct computer for this application. If you make more than one selection, then DCOM uses the first applicable one. Client applications may override your selections.

- [ ] Run application on the computer where the data is located.
- [ ] Run application on this computer.
- [x] Run application on the following computer:

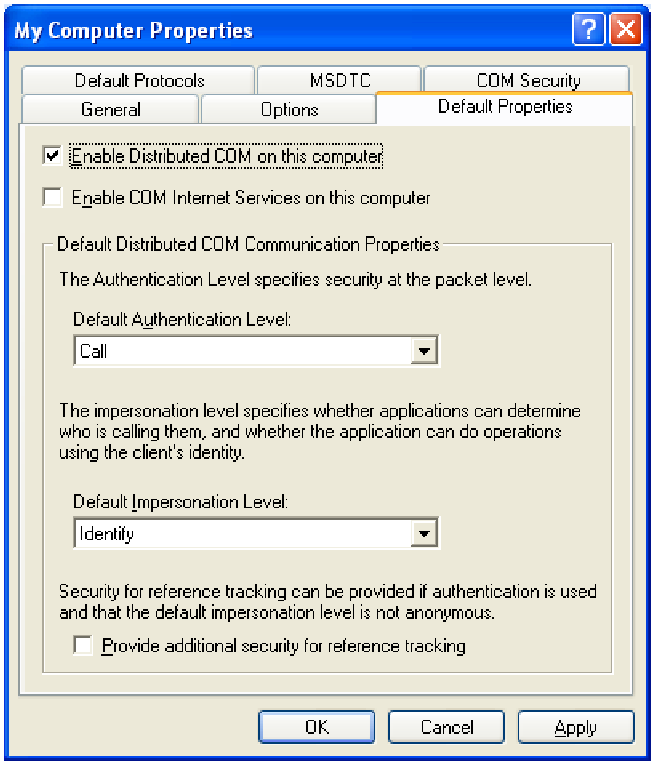
hostname or IP address

Choose **Apply** and then **OK**.

3 Select **My Computer** and choose the properties from the context menu.



The **My Computer Properties** dialog box will be displayed. Select the **Default Properties** tab.



[✓] Enable Distributed COM on this computer

[ ] Enable COM Internet Services on this computer

Default Distributed COM Communication Properties

The Authentication Level specifies security at the packet level.

Default Authentication Level: Call

The impersonation level specifies whether applications can determine who is calling them, and whether the application can do operations using the client's identity.

Default Impersonation Level: Identify

Security for reference tracking can be provided if authentication is used and that the default impersonation level is not anonymous.

[ ] Provide additional security for reference tracking

Choose the check box **Enable Distributed COM on this computer**, set the default authentication level to **Call** and the default impersonation level to **Identify**.

Choose **OK**.

Testing the Connection

You are now ready to test the connection between the client machine and the server machine.

Test the TCP/IP Connection

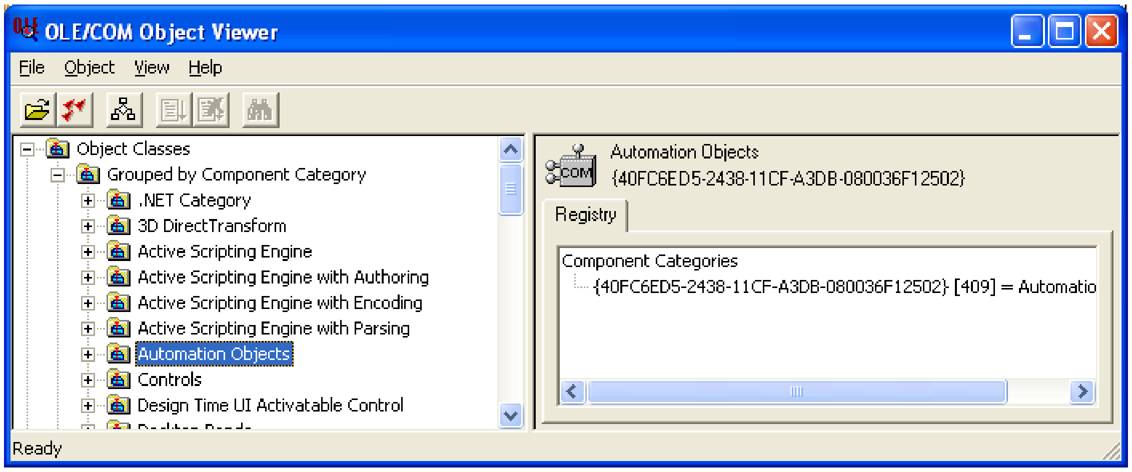
Test the TCP/IP connection between the client and the server (use, for example, ping).

Test the Remote Call

To test whether an application can be called remotely, you can use the OLE/COM Object Viewer:

Run the OLE/COM Object Viewer on the client.

The **OLE/COM Object Viewer** dialog box will be displayed:



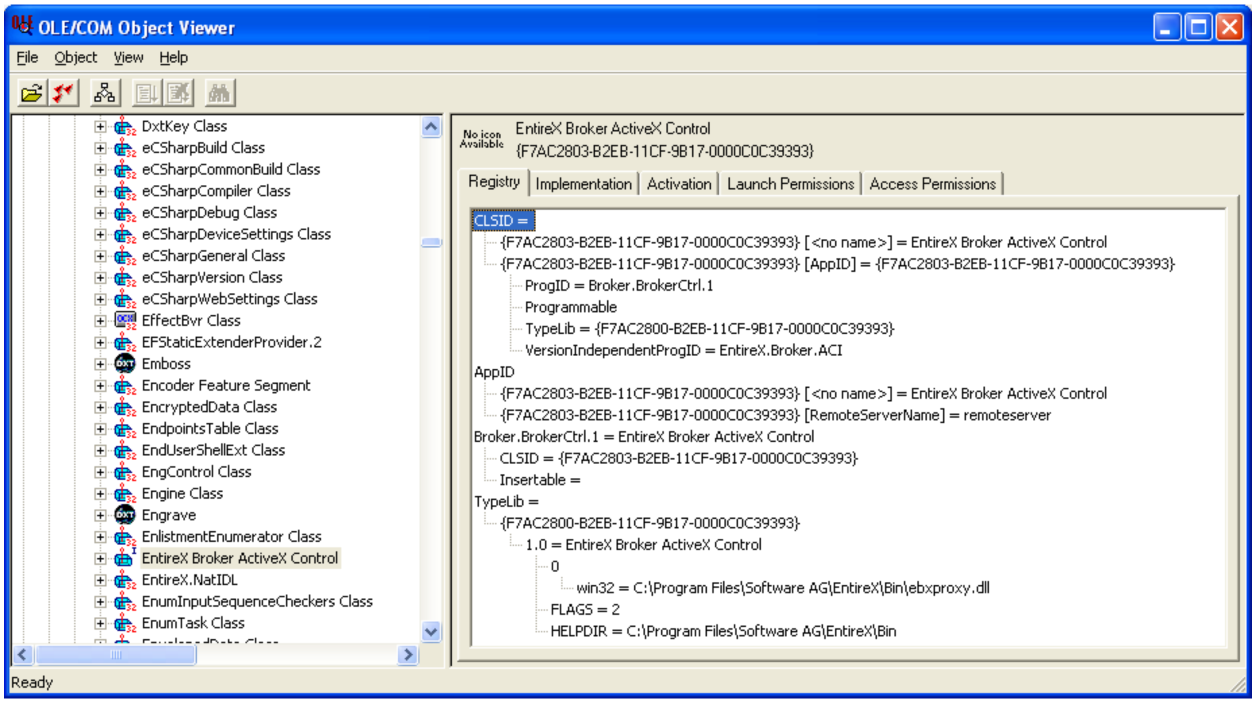
```
Object Classes
  Grouped by Component Category
    .NET Category
    3D DirectTransform
    Active Scripting Engine
    Active Scripting Engine with Authoring
    Active Scripting Engine with Encoding
    Active Scripting Engine with Parsing
    Automation Objects
    Controls
    Design Time UI Activatable Control
```

```
Automation Objects
{40FC6ED5-2438-11CF-A3DB-080036F12502}

Component Categories
  {40FC6ED5-2438-11CF-A3DB-080036F12502} [409] = Automatio
```

Select **Automation Objects** in the navigation frame to display a list of all the automation objects on the client machine.

A screen similar to the one displayed below will be displayed:



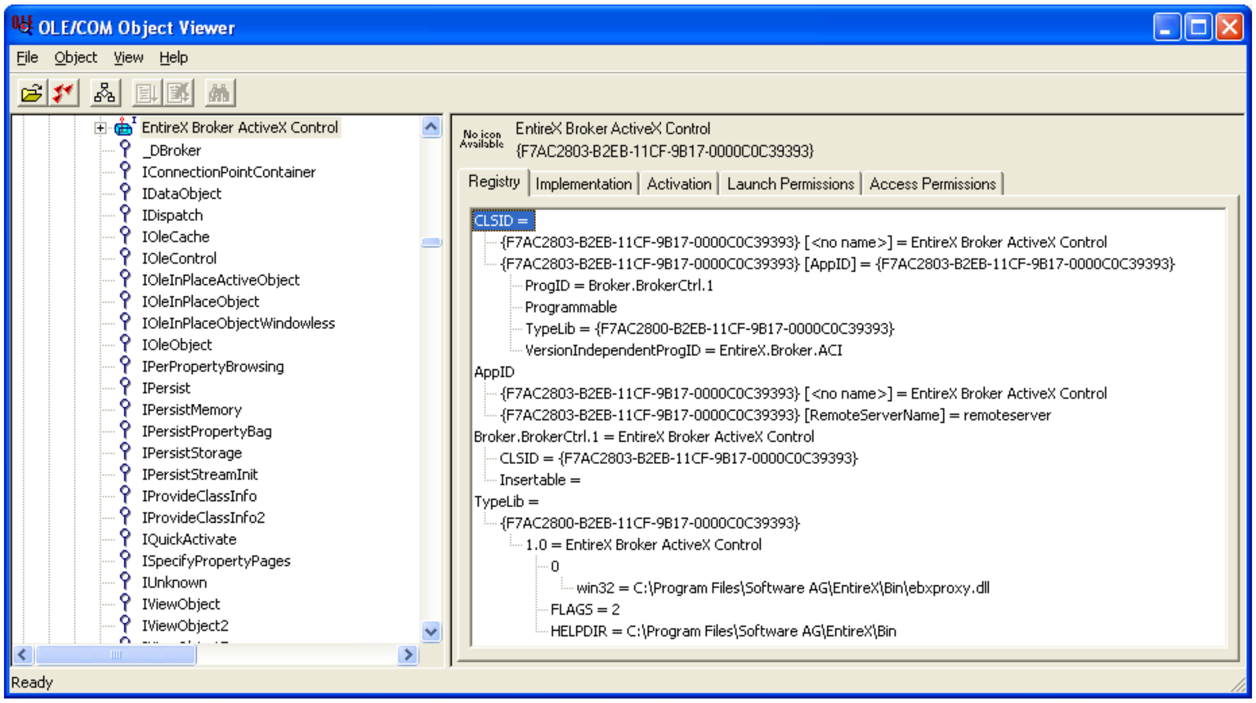
```
DxtKey Class
eCSharpBuild Class
eCSharpCommonBuild Class
eCSharpCompiler Class
eCSharpDebug Class
eCSharpDeviceSettings Class
eCSharpGeneral Class
eCSharpVersion Class
eCSharpWebSettings Class
EffectBvr Class
EFStaticExtenderProvider.2
Emboss
Encoder Feature Segment
EncryptedData Class
EndpointsTable Class
EndUserShellExt Class
EngControl Class
Engine Class
Engrave
EnlistmentEnumerator Class
EntireX Broker ActiveX Control
EntireX.NatIDL
EnumInputSequenceCheckers Class
EnumTask Class
```

```
EntireX Broker ActiveX Control
{F7AC2803-B2EB-11CF-9B17-0000C0C39393}

CLSID =
  {F7AC2803-B2EB-11CF-9B17-0000C0C39393} [<no name>] = EntireX Broker ActiveX Control
  {F7AC2803-B2EB-11CF-9B17-0000C0C39393} [AppID] = {F7AC2803-B2EB-11CF-9B17-0000C0C39393}
    ProgID = Broker.BrokerCtrl.1
    Programmable
    TypeLib = {F7AC2800-B2EB-11CF-9B17-0000C0C39393}
    VersionIndependentProgID = EntireX.Broker.ACI
AppID
  {F7AC2803-B2EB-11CF-9B17-0000C0C39393} [<no name>] = EntireX Broker ActiveX Control
  {F7AC2803-B2EB-11CF-9B17-0000C0C39393} [RemoteServerName] = remoteserver
Broker.BrokerCtrl.1 = EntireX Broker ActiveX Control
  CLSID = {F7AC2803-B2EB-11CF-9B17-0000C0C39393}
  Insertable =
TypeLib =
  {F7AC2800-B2EB-11CF-9B17-0000C0C39393}
    1.0 = EntireX Broker ActiveX Control
      0
        win32 = C:\Program Files\Software AG\EntireX\Bin\ebxproxy.dll
      FLAGS = 2
      HELPDIR = C:\Program Files\Software AG\EntireX\Bin
```

Select **EntireX Broker ActiveX Control**, open its context menu and choose **Create Instance**.

If the remote call is successful, the EntireX Broker component on the server machine will be called and the following screen will be displayed:



```
EntireX Broker ActiveX Control
  _DBroker
  IConnectionPointContainer
  IDataObject
  IDispatch
  IOleCache
  IOleControl
  IOleInPlaceActiveObject
  IOleInPlaceObject
  IOleInPlaceObjectWindowless
  IOleObject
  IPerPropertyBrowsing
  IPersist
  IPersistMemory
  IPersistPropertyBag
  IPersistStorage
  IPersistStreamInit
  IProvideClassInfo
  IProvideClassInfo2
  IQuickActivate
  ISpecifyPropertyPages
  IUnknown
  IViewObject
  IViewObject2
```

```
EntireX Broker ActiveX Control
{F7AC2803-B2EB-11CF-9B17-0000C0C39393}

CLSID =
  {F7AC2803-B2EB-11CF-9B17-0000C0C39393} [<no name>] = EntireX Broker ActiveX Control
  {F7AC2803-B2EB-11CF-9B17-0000C0C39393} [AppID] = {F7AC2803-B2EB-11CF-9B17-0000C0C39393}
    ProgID = Broker.BrokerCtrl.1
    Programmable
    TypeLib = {F7AC2800-B2EB-11CF-9B17-0000C0C39393}
    VersionIndependentProgID = EntireX.Broker.ACI
AppID
  {F7AC2803-B2EB-11CF-9B17-0000C0C39393} [<no name>] = EntireX Broker ActiveX Control
  {F7AC2803-B2EB-11CF-9B17-0000C0C39393} [RemoteServerName] = remoteserver
Broker.BrokerCtrl.1 = EntireX Broker ActiveX Control
  CLSID = {F7AC2803-B2EB-11CF-9B17-0000C0C39393}
  Insertable =
TypeLib =
  {F7AC2800-B2EB-11CF-9B17-0000C0C39393}
    1.0 = EntireX Broker ActiveX Control
      0
        win32 = C:\Program Files\Software AG\EntireX\Bin\ebxproxy.dll
      FLAGS = 2
      HELPDIR = C:\Program Files\Software AG\EntireX\Bin
```

If you receive an error message (for example “Class not registered”) check the following:

- the TCP/IP connection (with PING)
- the security definitions on the server with **Component Services**
- the remote server name on the client (this can also be checked with the OLE/COM Object Viewer)

When the connection has been established, you will be able to run your application on the client. Note that Broker ActiveX Control must be used as automation server. For information on how to use Broker ActiveX Control with Visual Basic see ***Using Broker ActiveX Control as an Automation Server***.

9 Reference

- Methods 62
- Properties 63

Methods

This section describes the methods of Broker ActiveX Control.

Broker ACI

The following methods are useful for writing applications using the native interface.

Method	Description
`BSTR GetReceiveData()`	Return the received data inner string
`BSTR GetErrorText()`	Return the last received error message.
`BOOL SetSendDataLong(String, Long)` or `BOOL SetSendData (String, Short)`	Copy user's data buffer into the send buffer.
`BOOL InvokeBrokerFunction()`	Invoke the broker function call. Set the properties `Function` and `Option`.

Transaction Objects

Method	Description
`Bool DefineTOMethods(String)`	Starts the TO editor. If you specify a valid TOR name, this TO is then loaded into the editor. If a valid TOR name is not specified, the currently loaded TO will be displayed or an empty editor will be started.
`Bool LoadTransObject(String)`	Loads and initializes a transaction object. You must specify a valid TOR file name; otherwise FALSE will be returned.
`Object CreateTransObject(String)`	Loads and initializes a transaction object. You must specify a valid TOR file name. An object reference will be returned, which can be used to call the methods defined in the TO. If loading fails, a null reference will be returned.
`Object CreateTransObjectSA(String)`	This method uses the safe array implementation for arrays instead of the collection implementation. If you experience problems accessing arrays with an automation controller, try using this method to instantiate a TOR object.

Properties

Most properties of Broker ActiveX Control correspond to the Broker ACI fields. The properties must be set to the appropriate values before using any function.

If transaction object repository (TOR) files are used, it will not be necessary to set all the properties. See *Transaction Objects in Broker ActiveX Control*. The properties can also be supplied by means of the property pages (see *Using the Property Pages*).

Property Name	Broker ACI Field	Format	Length	API Version	Description
`AdapterError`	not used	String	8	2	
`AdCount`	not used	Long		2	
`APIVersion`	`API-VERSION`	Short		2	Possible values: 1, 2, 3, 4, 5, 6, 7, 8, 9. The default is 2. This value can be changed dynamically by setting the property. If the current value of the `Function` or `Option` property requires a minimal API version, the value of `APIVersion` will be adjusted automatically.
`BrokerID`	`BROKER-ID`	String	32	1	Target Broker ID. See *Using the Broker ID in Applications* and details on TCP/IP in *Transport Methods* in the ACI Programming documentation.
`BrokerSecurity`	`KERNELSECURITY`	String	1	7	
`ClientUserid`	`CLIENT-UID`	String	32	2	The partner's user ID.
`CommitTime`	`COMMITTIME`	String	17	7	Readonly property. Time when UOW was committed. Format: YYYYMMDDHHMMSSms ms = milliseconds in Possible Values field.
`CompressLevel`	`COMPRESSLEVEL`	String	1	7	Compression level. Possible values: N/Y/0-9. The first character of the string will be used as the compression value. If you type YES, the

Property Name	Broker ACI Field	Format	Length	API Version	Description
					character Y will be used and ES will be cut off. Example: Broker1.CompressLevel = "6". See also *Data Compression*.
`ConvID`	`CONV-ID`	String	16	1	Conversation ID, see *Managing Conversation Contexts*.
`ConvStatus`	`CONV-STAT`	Short		2	Contains the status of the conversation when the RECEIVE function is complete. See *Managing Conversation Contexts*. Possible values: 1 NEW 2 OLD 3 NONE
`EncryptionLevel`		Short		6	Deprecated. For encrypted transport we strongly recommend using the Secure Sockets Layer/Transport Layer Security protocol. See *SSL/TLS, HTTP(S), and Certificates with EntireX* in the platform-independent Administration documentation.
`Environment`	`ENVIRONMENT`	String	32	1	Pass additional information to *Translation User Exit*. For more information see ACI `ENVIRONMENT`.
`ErrorCode`	`ERROR-CODE`	String	8	1	Broker error code, see *Error Handling*.
`ErrorMsg`	not used	String	40	1	Contains the error message to the corresponding error code.
`ForceLogon`	`FORCE-LOGON`	Boolean		6	Possible values: Y, N.
`Function`	`FUNCTION` Possible values: 1 `SEND` 2 `RECEIVE` 4 `UNDO` 5 `EOC` 6 `REGISTER`	Short		1	The functions to be performed by Broker.

Property Name	Broker ACI Field	Format	Length	API Version	Description
	7 `DEREGISTER` 8 `VERSION` 9 `LOGON` 10 `LOGOFF` 13 `SYNCPOINT` 14 `KERNELVERS`				
`localeString`	`LOCALE-STRING`	String	40	4	The Broker ActiveX Control uses the Windows ANSI codepage to convert the Unicode (UTF-16) representation within BSTRINGS to the encoding sent to or received from the broker. This codepage is also transferred as part of the locale string to tell the broker the encoding of the data. It is not possible to use any codepage other than the codepage configured for Windows in the Regional Settings. If you want to adapt the Windows ANSI codepage, see the Regional Settings in the Windows Control Panel and your Windows documentation. Enable character conversion in the broker by setting the service-specific attribute `CONVERSION` to "`SAGTCHA`". See also *Configuring ICU Conversion* under *Configuring Broker for Internationalization* in the platform-specific Administration documentation. More information can be found under *Internationalization with EntireX*.
`MessageId`	not used	String	32	2	
`MessageType`	not used	String	32	2	
`NewPassword`	`NEWPASSWORD`	String	32	2	

Property Name	Broker ACI Field	Format	Length	API Version	Description
`Option`	`OPTION` Possible values: 0 NULL 1 `MSG` 2 `HOLD` 3 `IMMED` 4 `QUIESCE` 5 `EOC` 6 `CANCEL` 7 `LAST` 8 `NEXT` 9 `PREVIEW` 10 `COMMIT` 11 `BACKOUT` 12 `SYNC` 13 `ATTACH` 14 `DELETE` 15 `EOCCANCEL` 16 `QUERY` 17 `SETUSTATUS` 18 `ANY` 19 no longer used 20 no longer used 21 `CHECKSERVICE`	Short		1	
`Password`	`PASSWORD`	String	32	1	
`ReceiveBufferLength`	`RECEIVE-LENGTH`	Long		3	Length of the receive buffer.
`ReceiveBufferSize`	`RECEIVE-LENGTH`	Short		1	This is an old property. Can be used instead of `ReceiveBufferLength` - for buffers with less than 32 KB only.
`ReturnDataLength`	`RETURN-LENGTH`	Long		3	Length of returned data.
`ReturnLength`	`RETURN-LENGTH`	Short		1	This is an old property. Can be used instead of `ReturnDataLength` - for buffers with less than 32 KB only.

Property Name	Broker ACI Field	Format	Length	API Version	Description
`SecurityToken`	`SECURITY-TOKEN`	String	32	2	This is handled automatically, but can be filled in by the user if required.
`SendBufferSize`		Short		1	No longer used.
`ServerClass`	`SERVER-CLASS`	String	32	1	These three Broker parameters form the target service.
`ServerName`	`SERVER-NAME`	String	32	1	
`Service`	`SERVICE`	String	32	1	
`Store`	`STORE`	Short		2	Possible values: 0 NULL 1 OFF 2 BROKER
`Token`	`TOKEN`	String	32	1	
`UOWID`	`UOWID`	String	16	3	
`UOWStatus`	`UOWSTATUS` Possible values: 0 NONE 1 RECEIVED 2 ACCEPTED 3 DELIVERED 4 BACKEDOUT 5 PROCESSED 6 CANCELLED 7 TIMEOUT 8 DISCARDED 9 FIRST 10 MIDDLE 11 LAST 12 ONLY	Short		3	
`UOWStatusPersist`	`UOW-STATUS-PERSIST`	Short		3	
`UOWTime`	`UWTIME`	String	8	3	
`UserData`	`USER-DATA`	String	16	2	This field is not converted by the Broker. If the field contains H'00', only the data up to the first H'00' will be sent.
`UserID`	`USER-ID`	String	32	1	User ID.

Property Name	Broker ACI Field	Format	Length	API Version	Description
`UserStatus`	`USTATUS`	String	32	3	
`Wait`	`WAIT`	String	8	1	Possible values: Yes No <n>S - waiting n Seconds (max 99999) <n>M - waiting n Minutes (Max 99999) <n>H - waiting n Hours (max 99999). See *Blocked and Non-blocked Broker Calls*.
`UOWStatusLife`		String	8	8	