.NET Wrapper Reference

This chapter covers the following topics:

- Attributes
- Classes

Attributes

Attribute classes are defined and implemented in the .NET Wrapper runtime and used in the C# client stub code to hold information extracted from the IDL file.

EntireXVersionAttribute

This attribute contains version information.

Example

```
[EntireXVersion("9.7.0.n")]
public class Example
```

LibraryAttribute

This attribute contains the library name.

Example

```
[Library("EXAMPLE")]
public class Example
```

BrokerAttribute

This attribute contains the Broker ID.

Example

```
[Broker("localhost:1971")]
public class Example
```

ServiceAttribute

This attribute contains the service name.

Example

```
[Service("RPC/SRV1/CALLNAT")]
public class Example
```

ProgramAttribute

This attribute contains the program name.

Example

```
[Program("CALC")]
public int Calculator(
  [SendAs(IdlType.A, Length=1f)][In] string operation,
  [SendAs(IdlType.I4)][In] int operand1,
  [SendAs(IdlType.I4)][In] int operand2
)
```

SendAsAttribute

This attribute contains type, length (fixed or dynamic) and dimension (fixed or dynamic) information.

Direction Attributes (In, Out)

These attributes contain direction information. They are supported natively by C#.

Example

```
[Program("HELLO")]
public void Hello(
  [SendAs(IdlType.A, Length=80f)][In] string client,
  [SendAs(IdlType.A, Length=80f)][In, Out] ref StringBuilder mail
)
```

Classes

The .NET Wrapper runtime defines and implements several generic service classes that are used in the generated C# client stub and by .NET client applications.

BigNumeric

Implementation of decimal values without upper and lower limit and a default number of 99 digits after the decimal sign.

Constructors

```
public BigNumeric (  BigNumeric number )
```

Copy Constructor.

```
public BigNumeric (  decimal number )
```

Constructor translating a decimal number into a `BigNumeric` value.

```
public BigNumeric (  Int32 number )
```

Constructor translating an Int32 number into a `BigNumeric` value.

```
public BigNumeric (  Int64 number )
```

Constructor translating an Int64 number into a `BigNumeric` value.

```
public BigNumeric (  double number )
```

Constructor translating a double to its exact `BigNumeric` representation.

```
public BigNumeric (  string number )
```

Constructor converting the string representation of a number into a `BigNumeric` object. The string representation consists of an optional sign, ’+’ or ’-’, followed by a sequence of zero or more decimal digits, optionally followed by a fraction which consists of a decimal point followed by zero or more decimal digits and optionally followed by an exponent which consists of the character ’e’ or ’E’ followed by one or more decimal digits. The value must not exceed the size of the N(U)/P(U) data type.

```
public BigNumeric (  string number , CulturInfo info )
```

Constructor converting the string representation of a number into a `BigNumeric` object. The desired culture info is used to interpret the numeric string.

Methods

```
public int GetHashCode()
```

Returns the hash code of the instance.

```
public string ToString()
```

Converts the value of the instance to its string representation.

```
public string ToString( NumberFormatInfo info )
```

Converts the value of the instance to its string representation using the desired number format.

```
public BigInteger Truncate()
```

Returns the integral part of the number.

```
public BigNumeric Truncate (  int scale )
```

Returns a new `BigNumeric` with a maximum number of digits after the decimal sign. The number is truncated to the desired precision if necessary.

```
public BigNumeric Round (  int scale )
```

Returns a new `BigNumeric` with a maximum number of digits after the decimal sign. The number is rounded to the desired precision if necessary. Rounding is performed according to the rounding mode "`HALF_UP`" of Java class `BigDecimal`.

```
public bool isNegative()
```

Returns true for negative numbers.

```
public static BigNumeric Random (  int preDecimal , int postDecimal )
```

Create a random `BigNumeric` number with the desired number of predecimal and a maximum of 99 postdecimal digits.

Operators

```
public static BigNumeric operator + ( BigNumeric operand1 , BigNumeric operand2 )
```

Adds two `BigNumerics`. The result inherits the scale of operand1.

```
public static BigNumeric operator - ( BigNumeric operand1 , BigNumeric operand2 )
```

Subtracts two `BigNumerics`. The result inherits the scale of operand1.

```
public static BigNumeric operator * ( BigNumeric operand1 , BigNumeric operand2 )
```

Multiplies two `BigNumerics`. The result inherits the scale of operand1.

```
public static BigNumeric operator / ( BigNumeric operand1 , BigNumeric operand2 )
```

Divides two `BigNumerics`. The result inherits the scale of operand1.

```
public static implicit operator BigNumeric ( decimal value )
```

Converts a decimal value to a `BigNumeric`.

```
public static implicit operator BigNumeric ( Int32 value )
```

Converts an int value to a `BigNumeric`.

```
public static implicit operator BigNumeric ( Int64 value )
```

Converts a long value to a `BigNumeric`.

```
public static explicit operator Decimal ( BigNumeric value )
```

Converts the `BigNumeric` to a decimal, throws an exception if the value doesn't match the decimal's numeric range.

```
public static bool operator < ( BigNumeric left , BigNumeric right )
```

Compares the value of two `BigNumerics`.

```
public static bool operator <= ( BigNumeric left , BigNumeric right )
```

Compares the value of two `BigNumerics`.

```
public static bool operator > ( BigNumeric left , BigNumeric right )
```

Compares the value of two `BigNumerics`.

```
public static bool operator >= ( BigNumeric left , BigNumeric right )
```

Compares the value of two `BigNumerics`.

```
public static bool operator == ( BigNumeric left , BigNumeric right )
```

Compares the value of two `BigNumerics`.

```
public static bool operator != ( BigNumeric left , BigNumeric right )
```

Compares the value of two `BigNumerics`.

```
bool Equals ( object o )
```

Compares the desired object with the value of this.

```
public static bool operator < ( BigNumeric left , Decimal right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator <= ( BigNumeric left , Decimal right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator > ( BigNumeric left , Decimal right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator >= ( BigNumeric left , Decimal right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator == ( BigNumeric left , Decimal right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator != ( BigNumeric left , Decimal right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator < ( Decimal left , BigNumeric right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator <= ( Decimal left , BigNumeric right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator > ( Decimal left , BigNumeric right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator >= ( Decimal left , BigNumeric right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator == ( Decimal left , BigNumeric right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

```
public static bool operator != ( Decimal left , BigNumeric right )
```

Compares the value of a `BigNumeric` with the value of a decimal.

Properties

```
public static BigNumeric Zero
```

Returns a `BigNumeric` representing the value 0

```
public static BigNumeric One
```

Returns a `BigNumeric` representing the value 1

```
public static BigNumeric MinusOne
```

Returns a `BigNumeric` representing the value -1

```
public int Scale
```

Set/Get maximum number of digits after decimal sign

Broker

This class represents an EntireX Broker session and handles the connections to the Broker.

Constructors

```
public Broker()
```

Default Broker for default user.

The values for the default Broker and user are taken in the following order

- from the application’s configuration file or
- from the [Broker] attribute of the client stub (Broker values only) or
- from hard-coded constants `localhost:1971` and `ERX-USER`.

```
public Broker(string hostName)
```

Broker on `hostName` for default user (`ERX-USER`).

```
public Broker(string hostName, string userName)
```

Broker on `hostName` for `userName`.

```
public Broker(string hostName, string userName, string token)
```

Broker on `hostName` for `userName` with `token`.

Methods

```
public void Logon()
```

Performs a logon to the Broker with the default user ID and password (that were set, for example, with the `UserID` and `Password` property).

```
public void Logon(string password)
```

Performs a logon to the Broker with the given password.

```
public void Logon(string password, string newPassword)
```

Performs a logon to the Broker with the given password and changes the `password` to `newPassword`

```
public void Logon(string userID, string token, string password)
```

Performs a logon to the Broker with the given user ID, token and password

```
public void Logoff()
```

Performs a logoff from the Broker.

Properties

```
public bool ForceLogon
```

Specifies whether force logon is performed. The default is false.

```
public char BrokerSecurity
```

Sets or retrieves the level and type of Broker security to be used.

’N’ : no security
’Y’ : default EntireX Security
’C’ : user-specific security

```
public int CompressionLevel
```

Specifies what compression level should be used. Possible values are in the range 0 to 9.
The following values have a dedicated purpose.

0: do not compress
1: use compression method with best speed
6: use default compression
8: deflated
9: use best compression
The default value is 0 (no compression)

```
public int EncryptionLevel
```

Specifies what encryption level should be used. Possible values are:

0: no encryption
1: encrypt communication with the Broker
2: encrypt communication with the RPC server
The default value is 0.

```
public string BrokerID
```

Retrieves the Broker ID of the given Broker class instance. This property is read-only.

```
public byte[] IAFToken
```

Sets or retrieves the IAF token of a given Broker instance.

```
public string Password
```

Sets the password of a given Broker class instance for subsequent authentication. This property is write-only.

```
public byte[] SecurityToken
```

Sets or retrieves the security token of a given Broker class instance. The default value is null.

```
public string Token
```

Sets or retrieves the token of the given Broker class instance. The default value is null.

```
public string UserID
```

Sets or retrieves the user ID of the given Broker class instance for subsequent authentication.

```
public string ApplicationName
```

The application name used for the client calls. Max. 64 characters. Set this property before calling one of the Logon methods if you want to replace the default application name.

Deprecated Properties

```
public Compress Compression
```

Please use the `CompressionLevel` property instead.

```
public bool Encryption
```

Please use the `EncryptionLevel` property instead.

Example

```
Broker broker = new Broker("ibm2:3762", "ERX-USER");
broker.Logon("ERX-PASS");
```

Service

Constructors

```
public Service()
```

Default service with default Broker.

```
public Service(string libraryName)
```

Service for given library with default Broker.

```
public Service(Broker broker)
```

Service for given Broker.

```
public Service(Broker broker, string trinity)
```

Service for given Broker and service name: `class/server/service` (for example `RPC/SRV1/CALLNAT`).

```
public Service(string Broker broker, string trinity, libraryName)
```

Service for given Broker, service name: `class/server/service` and library.

Methods

```
public int SetReliableState(int uReliableState)
```

Set the Reliable State. Possible values:

`RELIABLE_OFF (0)` - default value
`RELIABLE_AUTO_COMMIT (1)`
`RELIABLE_CLIENT_COMMIT (2)`

See *Reliable RPC for .NET Wrapper*.

```
public int ReliableCommit()
```

Do a commit in Reliable State `RELIABLE_CLIENT_COMMIT`.

```
public int ReliableRollback()
```

Do a rollback in Reliable State `RELIABLE_CLIENT_COMMIT`.

```
public int GetReliableID(ref StringBuilder ReliableID)
```

Get the `ReliableID`.

```
public int GetReliableStatus(StringBuilder ReliableID, ref StringBuilder ReliableStatus)
```

Get the Reliable Status. Possible values:

```
RECEIVED
ACCEPTED
DELIVERED
BACKEDOUT
PROCESSED
CANCELLED
TIMEOUT
DISCARDED
```

See *Broker ACI Fields* for more information.

```
public void CloseConversation()
```

Close an RPC conversation.

```
public void CloseConversationCommit()
```

Close an RPC conversation and commit.

```
public void UserIDAndPassword(string user, string password)
```

Specify user ID and password for a service.

```
public void OpenConversation()
```

Open an RPC conversation.

```
public unsafe object Invoke ( string library , string method , params object[] objArray )
```

where	library	is the name of the class in the generated client stub
	method	the name of the method to be invoked
	objArray	the methods parameters as an array of objects - the array size must fit the parameter count of the method .

`Invoke` returns the result (if any) of the invoked method.

The initialisation of the parameter array follows the rules:

1. Parameters of type groups, structs and arrays have to be assigned as follows

   ```
   int[] numbers = new int[10] ;
   ...
   objArray[i] = numbers ;
   ```

2. [in,out] and [out] parameters of the simple data types bool, char, byte, sbyte, decimal, float, double, short, int and DateTime have to be assigned as follows:

   ```
   int number = 4711 ;
   ...
   objArray[i] = new Ref ( ref number )  ;
   ```

   where `Ref` is the class `SoftwareAG.EntireX.NETWrapper.Runtime.Ref`.

Note:
The name of the class and the assembly name (file name) have to be identical. For each class, a separate assembly is required. All these assemblies have to be placed in the folder of the client executable or have to be configured according to the rules described in *Configuring the EntireX RPC Server for use with the .NET Framework*.

Properties

```
public Encoding CharacterEncoding
```

Define an encoding for character translation. Default is `System.Text.Encoding.GetEncoding(0) (current locale)`. See also the .NET Framework class library documentation for `System.Text.Encoding`.

```
public bool Encryption
```

Specify whether encryption is used. The default is false.

```
public bool NaturalLogon
```

Specify whether Natural logon should be performed. The default is false. If `NaturalLogon` is set to true but no `RPCUserID` and `RPCPassword` have been defined, the runtime uses the Broker user ID and password (provided the Broker password has been set with the `Password` property).

```
public Broker Broker
```

Sets or retrieves the Broker instance associated with the given Service instance.

```
public string RPCUserID
```

Sets or retrieves the RPC user ID of a given Service instance.

```
public string RPCPassword
```

Sets the RPC user password of a given Service instance.

```
public string ServerAddress
```

Retrieves the server address (class/server/service triplet) of a given Service instance.

```
public string Library
```

Sets or retrieves the library name of a given Service instance.

```
public Uint Timeout
```

Sets or retrieves the timeout value for a given Service instance. `Timeout = 0` is invalid. If 0 is set, a default of 50 seconds will be used.

Example

```
Service service = new Service( broker, "RPC/SRV1/CALLNAT", "EXAMPLE");
service.UserIDAndPassword("RPC-USER", "RPC-Password");
```

XException

Properties

```
public int errorCode
```

If an `XException` is thrown, `errorCode` contains the specific error code.

```
public string Message
```

If an `XException` is thrown, `Message` contains the specific error message. See *Message Class 2002 - .NET Wrapper*.

Example

```
try {
      ...
    } catch (EntireX.XException e) {
      Console.WriteLine( e.Message ) ;
    };

    Output: "02150148: EntireX Broker not active.
```