Using ActiveX with Caché

Version 2017.2
2020-06-25

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

Using ActiveX with Caché
Caché Version 2017.2 2020-06-25

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Figures

List of Tables

About This Book

The Caché ActiveX binding provides access to Caché from any application that supports ActiveX objects.

This book contains the following sections:

- Introduction
- The Elements of the ActiveX Interface
- Configuring a Visual Basic Project
- The Parts of a Caché Objects/Visual Basic Application
- Caché ActiveX API Reference

There is also a detailed Table of Contents.

For general information, see *Using InterSystems Documentation*.

1

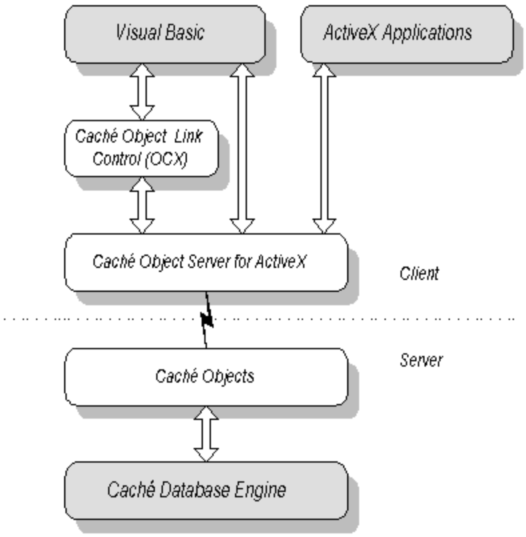
Introduction

This document covers:

- The Elements of the ActiveX Interface
- Configuring a Visual Basic Project
- The Parts of a Caché Objects/Visual Basic Application

Caché objects provide connectivity with a wide variety of client applications and development tools via an ActiveX interface:

Figure 1–1: Caché Object Server for ActiveX



Caché objects include the following ActiveX components:

- Caché Object Server for ActiveX — An ActiveX automation server that exposes Caché objects as ActiveX objects.
- Caché List Control — An ActiveX control written for Visual Basic that aids in the display of query results. You must provide the interface for selecting queries and query parameters for execution.
- Caché Query Control — An ActiveX control written for Visual Basic that provides a simple interface for executing queries and displaying the results. The Caché Query Control provides an interface for selecting at runtime any query that returns the ID and for specifying any query parameters.
- Caché Object Form Wizard — A Visual Basic add-in that allows you to quickly and easily create a simple form to access properties of a single Caché class.

2

The Elements of the ActiveX Interface

The Caché ActiveX binding provides access to Caché from any application that supports ActiveX objects. This binding allows you to instantiate and manipulate Caché objects within client applications, and provides a transparent interface to their server-side object instantiations. Examples in this book are written in Visual Basic, but the Caché ActiveX binding can be used just as easily in many other development environments.

2.1 The Caché Object Server for ActiveX

The Caché Object Server for ActiveX (CacheActiveX.dll) is a complete ActiveX in-process automation server that gives client applications access to server-based Caché objects. Internally, the Caché Object Server for ActiveX creates an ActiveX proxy object that mirrors an object on a Caché server. Object properties are exposed as ActiveX properties. Similarly, methods are exposed as ActiveX methods. When a method of an ActiveX proxy object is called, it causes the corresponding method to be called by the mirrored Caché object on the server machine.

Every client-side **ObjInstance** proxy object refers to a Caché object on a server. It is possible for several client-side objects to refer to the same server-side object. With each added reference, the server-side object's reference count is increased by one as long as any client-side objects are connected to it.

Client-side objects that perform special functions, such as the SysList and ResultSet objects, are not based on a Caché object and do not refer to an object on a server. See the Caché ActiveX API Reference for more information on the types of objects available from ActiveX.

The client-side proxy object is an ActiveX object and is referred to using a pointer to the IDispatch interface. For example, to access the properties and methods of a patient object from Visual Basic, use the following code:

```
Print patient.Name
'returns the value of Name for the referenced patient object
Print patient.Admit
'runs the Admit() method of the patient object on the server
```

As an improvement on typical ActiveX automation servers, the Caché Object Server for ActiveX does not require you to register every object class that you define with the client operating system. Instead, it determines class information at run time. This lets you develop large applications with many client machines and many server-based objects without having to maintain ActiveX registry entries.

2.2 Upgrading from CacheObject.dll

Although CacheActiveX.dll has been the preferred version of the Caché ActiveX binding since Caché 5.1, the binding is also available through an older DLL, CacheObject.dll. This DLL is slower and has some small differences, as noted in this documentation. You should use CacheActiveX.dll for all new development. The older DLL is provided only for backward compatibility. The two DLLs differ primarily in the following ways:

- CacheActiveX.dll can be used on 64-bit Windows in a 64-bit process.
- Internally, CacheActiveX.dll uses the Caché C++ binding to get object-level access to a Caché server, providing better performance in some cases due to more sophisticated object caching.
- CacheActiveX.dll supports Caché security (including features such as the ability to use Kerberos authentication), and does not support the less secure encrypted password feature used by CacheObject.dll.
- The default SQL runtime display mode for CacheActiveX.dll is `ODBC`, rather than `Logical` as inCacheObject.dll.
- When using CacheActiveX.dll proxy objects (see The Objinstance Class), the **sys_Close()** method actually closes the server object. The method does nothing when called by CacheObject.dll proxy objects.

To preserve complete compatibility with existing applications, Caché installs both CacheActiveX.dll and CacheObject.dll. By default, existing applications will continue to use the original CacheObject.dll. If you wish to use the newer binding, you must modify your existing application to reference this new DLL, and test that your application performs as expected. To upgrade your code from CacheObject.dll to CacheActiveX.dll, perform the following steps:

- Replace all references to **CacheObject.Factory** with **CacheActiveX.Factory**.
- Make any changes required by the switch from `Logical` to `ODBC` SQL runtime display mode (see the ResultSet.SetRunTimeMode() method).
- Optionally, modify connection logic to use the new Caché security features, as described in Connecting to a Server.

3

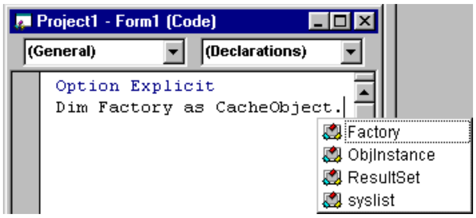
Configuring a Visual Basic Project

Although you can use Caché objects from a Visual Basic project without any configuration, you must configure your Visual Basic project if you want to use early binding. Early binding is a feature of Visual Basic that allows you to declare objects so they carry information about their type. For example, with early binding, you can specify:

```
Dim MyList as CacheActiveX.SysList
```

Early binding also provides pop-up menus which show you possible elements to use to finish a statement. For example, you can choose from a list of possible object types to associate with a new instance:

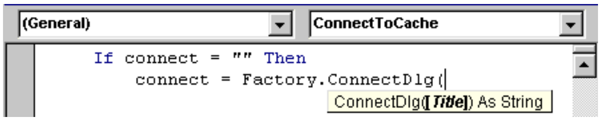
Figure 3–1: Automatic Continuation in Visual Basic Code



```
Option Explicit
Dim Factory as CacheObject.
```

Early binding also provides pop-up syntax windows for Caché elements:

Figure 3–2: Pop-Up Help Window in Visual Basic



```
    If connect = "" Then
        connect = Factory.ConnectDlg(
                      ConnectDlg([Title]) As String
```

To use early binding, select the project references as outlined below. The Visual Basic project configuration process consists of selecting the project references. Whenever you create a new Visual Basic project, you must add CacheActiveX.dll to your project references. To do this:

1. Within Visual Basic, open the references window using the **References** menu item in the **Project** menu.
2. Select the check box to the left of this DLL.

If this DLL does not exist in a list of available references:

1. Click on the **Browse** button.
2. Find CacheActiveX.dll in your Program Files\Common Files\Intersystems\Cache directory and double-click it.

Note: If you are working with legacy code and must use the older DLL, CacheObject.dll, you can add that to your project references in a similar way. CacheObject.dll is located in the <installation-root>\bin directory.

4

The Parts of a Caché Objects/Visual Basic Application

This section describes the basic actions in Visual Basic that manipulate Caché objects. It covers:

- Connecting to a Server
- Creating a New Object Instance
- Saving an Object
- Opening an Existing Object
- Using Caché Objects in Visual Basic
- Using Callback Functionality in Visual Basic
- Running a Query in Visual Basic
- Using Streams in Visual Basic
- Using Lists in Visual Basic
- Error Trapping in Visual Basic

4.1 Connecting to a Server

Typically, a client application first creates a CacheActiveX.Factory object and establishes a connection to a Caché server. Within Visual Basic the following code provides one way to do this:

```
'Visual Basic Code
option Explicit
Dim factory As CacheActiveX.Factory

Private Sub Form_Load()

' Create instance of factory object
' You must use "Set" to assign an object value
Set factory = CreateObject("CacheActiveX.Factory")

' Establish connection to server if one doesn't exist
If Not factory.IsConnected() Then
     Dim connectstring As String
     ' You can explicitly specify the connection string:
     connectstring = "cn_iptcp:127.0.0.1[1972]:USER"

     ' Alternately, you can pop up a connection dialog
     ' This method returns the connection string.
     connectstring = factory.ConnectDlg()

     Dim success As Boolean
     success = factory.Connect(connectstring)
End If
End Sub
```

First, create an instance of a CacheActiveX.Factory object by calling Visual Basic's **CreateObject** function:

```
Set factory = CreateObject("CacheActiveX.Factory")
```

CreateObject fails if it is unable to find or otherwise load the DLL (which contains the implementation of the CacheActiveX.Factory object). See the chapter "Configuring a Visual Basic Project."

As in the code above, you can optionally test a CacheActiveX.Factory object to see if it has an established connection using the **factory.IsConnected** method. This method returns True if a connection exists.

Next, using the CacheActiveX.Factory object, you then establish a connection to a server by using the **Connect** method of the CacheActiveX.Factory object:

```
success = factory.Connect(connectstring)
```

Connect is passed a connection string consisting of a connection protocol, an IP address or Fully Qualified Domain Name (FQDN) and port number, and a namespace separated by colons. The connection protocol is always "cn_iptcp", signifying the TCP/IP protocol. The IP address and port together uniquely specify a specific Caché server. The port number is specified in square brackets ("[" and "]") immediately following the IP address or FQDN. The namespace specifies which namespace contains your Caché objects. If you omit the namespace, your application connects to the default namespace.

You can directly pass the **Connect** method a connection string or you can let your end users specify the connection information in a connection dialog box. This dialog pops up whenever the **ConnectDlg** method of the CacheActiveX.Factory object is executed. Once the end user specifies the connection information and clicks the **Okay** button, the **ConnectDlg** method returns the resulting connection string which can be passed to **Connect** to create the connection.

Connect returns True if it has successfully connected to a server and False if it fails.

4.1.1 Defining a Connection String

A *connectstring* has the following basic form:

```
cn_iptcp:ipaddress[port]:namespace
```

connectstring consists of the following parts:

- `cn_iptcp:` — A literal specifying the connection type. ActiveX only supports TCP/IP as the connection mechanism.
- *ipaddress* — The IP address or Fully Qualified Domain Name (FQDN).
- *port* — The port number for this connection, enclosed in square brackets and appearing immediately after the IP Address. Together, the IP address or FQDN and the port specify a unique Caché server.
- *namespace* — The namespace containing the Caché objects to be used. This namespace must have the Caché system classes compiled, and must contain the objects you want to manipulate from ActiveX.

For example, the **Connect** method can be used as follows:

```
' Establish connection to server
Dim connectstring As String
connectstring = "cn_iptcp:127.0.0.1[1972]:USER"

Dim success As Boolean
success = factory.Connect(connectstring)
```

4.1.1.1 Defining a Secure Connection

A secure *connectstring* uses connection type `scn_iptcp:` (rather than `cn_iptcp:`), and has the following basic form:

```
scn_iptcp:ipaddress[port]:namespace:srv_principal_name:security_level:username:password
```

If security is not enabled on the Caché server, all content after *namespace* is ignored. If security is enabled, *connectstring* must include the *srv_principal_name* and *security_level*. The *username* and *password* are optional arguments used with SSL.

The security arguments are defined as follows:

- `scn_iptcp:` — A literal specifying the connection type.
- *ipaddress* — The IP address or Fully Qualified Domain Name (FQDN).
- *port* — The port number for this connection, enclosed in square brackets and appearing immediately after the IP Address. Together, the IP address or FQDN and the port specify a unique Caché server.
- *namespace* — The namespace containing the Caché objects to be used. This namespace must have the Caché system classes compiled, and must contain the objects you want to manipulate from ActiveX.
- *srv_principal_name* — The Service Principal Name, which is the name of the principal that represents the Caché server. This is typically a standard Kerberos principal name, of the form `cache/machine.domain`, where cache is a fixed string indicating that the service is for Caché, *machine* is the machine name, and *domain* is the domain name, such as `intersystems.com`.
- *security_level* — An integer indicating Connection Security Level. Valid levels are:
  - 0 — Caché login (Password)
  - 1 — Kerberos (authentication only)
  - 2 — Kerberos with Packet Integrity
  - 3 — Kerberos with Encryption
  - 10 – SSL/TLS
- *username* — (Optional) The username under which the connection is being made.
- *Password* — (Optional) The password associated with the specified username.

For more information, see the “Authentication” chapter of the *Caché Security Administration Guide*.

4.1.1.2 Connection String for Use with the Older DLL

If you are using the older CacheObject.dll (see “Upgrading from CacheObject.dll”), the **Connect** method can accept a connection string that includes an encrypted password. (The newer CacheActiveX.dll does not need this, because this information is encoded by the ODBC driver.) That is, the connection string can have the following form:

```
cn_iptcp:ipaddress[port]:namespace:username:password
```

If Caché Direct security is turned on for the Caché server, there must be a password and it must be encrypted using the **EncryptPswd** function. In a project module, make the library available that includes the password encryption function:

```
Declare Function
        EncryptPswd Lib "CMVISM32" (ByVal pswd$, ByVal encrypt$) as Integer
```

Note: In Visual Basic, the declaration must appear on a single line.

Then make the calls that make the connection to the server:

```
' preallocate the buffer, 8 nulls
ret$ = String$(8,Chr(0))

' encrypt the password for the connection string
pwlen = EncryptPswd(PlainTextPassword,ret$)

' reset the length of the password string
' to match that of the encrypted password
ret$ = Left$(ret$,pwlen)

' Establish connection to server
Dim connectstring As String
connectstring = "cn_iptcp:127.0.0.1[1972]:USER:" & username & ":" & ret$

Dim success As Boolean
success = factory.Connect(connectstring)

If success <> True Then
    Dim MyMsg As String
    MyMsg = "Connection failed."
    MsgBox MyMsg
    End
End If
```

Note: If the Cache instance is installed with Normal security and `%SERVICE_CACHEDIRECT` is Unauthenticated (as required for usage of CacheObject.dll), UnknownUser must also be provided with additional role access (`%DEVELOPER` or `%MANAGER`). Otherwise, the connection attempt will fail with a `PROTECT` error. For more information, see "%Service_CacheDirect" in the "Services" chapter of the *Caché Security Administration Guide*.

4.2 Creating a New Object Instance

You can create a new instance of a Caché object by using the CacheActiveX.Factory object's **New** method:

```
Dim Patient As Object
Set Patient = factory.New("Patient")
```

The argument of **New** is the name of the Caché class to instantiate. **New** does the following:

- It creates a new instance of the specified object on the server. In this case, this is equivalent to calling

  ```
  Set object = ##class(Patient).%New()
  ```

  on the server. It returns the OREF value for this server-side object to the Caché Object Server for ActiveX.
- It creates a new instance of an ActiveX object on the client that is connected to the server-side object. It returns a handle (`LPDISPATCH`) to the ActiveX object.
- If **New** is unable to create the specified object, it raises an error condition (see "Error Trapping in Visual Basic").

4.3 Saving an Object

You can save an instance of a persistent object using its **%Save** method (called **sys_Save** in Visual Basic). Note that **sys_Save** is a method of a Caché object, not of the CacheActiveX.Factory object:

```
Dim status As String
patient.sys_Save
patient.sys_Close
Set patient = Nothing
```

Note that you must call **sys_Close** on an object when you are through using it. This closes the object on the server. In addition you should set patient to *Nothing* to close the object in Visual Basic.

Note also that there are no parentheses following the calls to **sys_Save** and **sys_Close**. This is because when a call is followed by empty parentheses, Visual Basic requires that its result be used. To ignore a call's return value (or if it has no return value), omit the parentheses, as in the calls above.

4.4 Opening an Existing Object

You can load an existing Caché object from the database by using the CacheActiveX.Factory object's **OpenId** method:

```
Dim Patient As Object
Set Patient = factory.OpenId("MyApp.Patient", id)
```

OpenId takes two arguments: the name of the Caché class to open and the ID value with which the object was stored in the database. The ID value is treated as a string. **OpenId** does the following:

- It loads the specified object into memory on the server. In this case, this is equivalent to calling:

  ```
  Set object = ##class(MyApp.Patient).%OpenId(id)
  ```

  on the server. It returns the OREF value for the in-memory server-side object to the ActiveX client.
- It creates a new instance of an ActiveX object on the client that is connected to the server-side object. It returns a handle to this ActiveX object.

If the **OpenId** method is unable to open the specified object, it raises an error condition (see “Error Trapping in Visual Basic”).

Note: There is also a client-side **Open** method that opens an object using its complete OID value (similar to the **%Open** method of the %Persistent class.

4.5 Using Caché Objects in Visual Basic

Once you have created an instance of a Caché object by calling the CacheActiveX.Factory object's **New** or **OpenId** methods you can use the object as you would any other Visual Basic object. For example, you can get and set property values:

```
Dim name As String

name = patient.Name
patient.Name = name
```

You can invoke methods on an object (note that methods are executed on the server):

```
patient.Admit()
```

There are some differences between using objects in Visual Basic and ObjectScript: Methods that start with “%” in Caché start with “sys_” in Visual Basic. Hence the ObjectScript **%Save** method is the **sys_Save** method in Visual Basic.

Also, you should not rely on the default property of Visual Basic objects. For example, use:

```
patient.Name = txtName.Text
```

instead of using:

```
patient.Name = txtName ' Don't use default property syntax
```

4.6 Using Callback Functionality in Visual Basic

The Caché Object Server for ActiveX supports callback functionality via the **SetOutput** method of the Factory class. This method allows you to direct system output to your client application. For example:

```
Set factory = CreateObject("CacheActiveX.Factory")
factory.SetOutput(Text1)
' System output now goes to textbox Text1
```

To disable output functionality you can call **SetOutput** with an empty string:

```
factory.SetOutput("")
```

4.7 Running a Query in Visual Basic

Caché provides a robust interface for performing queries. This interface can be accessed from ActiveX using the ResultSet object. ResultSet objects only exist in ActiveX; they have no corresponding objects in Caché. Each ResultSet object is tied to a specific query defined in a specific Caché class.

For example, run a query, ByName, in the Person class:

```
Dim rset As CacheActiveX.ResultSet
Dim columns As Integer
Dim counter As Integer
Set rset = factory.ResultSet("Person","ByName")

'Find out how many columns will be in the data
columns = rset.GetColumnCount()

'Now run the ByName query.
'This query will return all people
'whose names begin with the letter specified
rset.Execute("A")

'Cycle through the returned rows and access the data
While rset.Next()
     For counter = 1 To columns
         Print rset.GetData(counter)
     Next counter
Wend

'Close the result set
rset.Close()
```

Using this example, you can run the query ByName defined in the class Person. This query returns all people objects whose names begin with the letter specified (“A”). After executing the query, you can move from one row to the next by calling the **Next** method of the *rset* ResultSet object. You can use the ResultSet **GetColumnCount** method to find out how many columns are in each row of data, then use the **For** loop above to print the value stored in each column of the current row. The **While** loop moves you from one row to the next. Together, they allow you to print all of the data returned by the query. Once all of the data has been processed, you can use **Close** to close the ResultSet object.

4.8 Using Streams in Visual Basic

Stream properties are projected using the CacheActiveX.CharStream and CacheActiveX.BinaryStream classes. For example, suppose you have a Person object containing a character stream property, Memo. In addition to the standard stream methods **Read** and **Write**, you can use the client-side method **Data**, which fetches the entire stream in one operation:

```
' Visual Basic code
Dim person As Object
Dim memo As String1

' Open a Person object and copy its memo into a local variable
Set person=Factory.OpenId("Sample.Person",id)
memo=person.Memo.Data
```

The **GetPicture** method is a similar mechanism that efficiently copies binary data containing a bitmap image into a picture control. For example, if the Person object has a binary stream property, Picture, containing a bitmap image (such as a .jpg or .png file), you can display this image in Visual Basic as follows:

```
' Visual Basic code
Dim person As Object
Dim memo As String

' Open a Person object and show its picture in an Image control
Set person=Factory.OpenId("Sample.Person",id)
Image1.Picture = person.Picture.GetPicture
```

The **SetPicture** method can be used to copy an image in the other direction, from an Image control to a binary stream property:

```
' Visual Basic code
person.Picture.SetPicture Image1.Picture
```

4.9 Using Lists in Visual Basic

You can use CacheActiveX.SysList to pass a list (Caché $List format) back and forth between VB and Caché, as demonstrated in the following example:.

```
Dim colorList As New CacheActiveX.syslist
Dim ReturnData As Object
Dim myRef as Object

'populate the list with red,green,blue values
colorList.Add "red"
colorList.Add "green"
colorList.Add "Blue"

'create a new instance on the server
Set myref = o_factory.New("Pack.ClassA")
colorList2.Clear 'sets list to null

'pass a listbuild string and return the com object to ReturnData, which
'can then be used to get the items of the list. GetData()is an
'instance method that takes a %List as argument and returns %List.

set ReturnData = myref.GetData(colorList.Get)
Text1.Text = ReturnData.Item(1) 'display return string 'red'

colorList.sys_Close 'close object
Set colorList = Nothing 'close vb object
```

4.10 Error Trapping in Visual Basic

The CacheActiveX.Factory object raises a Visual Basic error if it encounters an error condition. You can obtain information about the error by using Visual Basic's *err* object. For example:

```
Private Sub OpenObject(id As String)

' Set up error handler
On Error GoTo errortrap:

' try to open non-existent object
Dim Patient As Object
Set Patient = factory.OpenId("Patient", id)

'......
Exit Sub

errortrap:
' handle error (show error string in dialog box)
MsgBox (Err.Description)

End Sub
```

Alternately, you can use the **GetErrorText** method of the CacheActiveX.Factory class. This method has the following syntax:

```
String GetErrorText(int ErrorNumber [, Param1, ..., Param10])
```

where *ErrorNumber* is a Caché error which was returned by the server and each of the *ParamN* arguments supplies a value to be substituted into the text of the error message. The return value appears in the language of the client's locale. For example:

```
' Caché was unable to save the class.
' The returned error is encapsulated by the error object,
'    #5659: "Property '%1' required"
msg = factory.GetErrorText(5659,"Name")
' msg now contains the string "Property 'Name' required"
```

The Caché ActiveX binding may return one of the following error codes:

Table 4–1: ActiveX Error Codes

Error Code	Meaning
9990	An error occurred while attempting to create an object. Check that the class exists in the specified namespace and server.
9991	An error occurred on the Caché system.
9992	An error occurred in a method with the return type %Status causing the status code to return False. See the Caché Error Reference for the list of built-in messages.

5

Caché ActiveX API Reference

Caché provides a set of ActiveX classes to manage the connection to a Caché server and interact with Caché objects. These classes are packaged within CacheActiveX.dll and include:

- **CacheActiveX.ObjInstance** — Each instance of this class is a client-side proxy object that mirrors a server-side Caché object. There is at least one client-side instance of CacheActiveX.ObjInstance for each server-side Caché object that a client application accesses.
- **CacheActiveX.Factory** — An instance of this class is used to establish a connection to a Caché server and create and manage Caché objects. An application only needs to create one instance of the CacheActiveX.Factory class.
- **CacheActiveX.SysList** — Instances of this class are used to create and manipulate data in Caché's $List format.
- **CacheActiveX.ResultSet** — Instances of this class are used to execute queries (either built-in class queries that are instances of %Library.Query or ad hoc SQL queries) and to process the results of these queries.
- **CacheActiveX.BinaryStream** — Instances of this class are used to create and manipulate Caché binary stream data.
- **CacheActiveX.CharStream**—Instances of this class are used to create and manipulate Caché character stream data.

Important: The Caché ActiveX Objects do not properly support the thousands separator in ActiveX numeric types for all locals. For example, the following statement (using Italian/Portuguese conventions for separators) would incorrectly treat both "." and "," as decimal symbols:

```
obj.MyCurrency = "123.457,99"  'Do not use!
```

5.1 The ObjInstance Class

Each CacheActiveX.ObjInstance object is a client-side representation of a server-based Caché object. There is at least one instance of ObjInstance for each server-based Caché object that a client application uses.

The **New()**, **Open()**, and **OpenId()** methods of the **CacheActiveX.Factory** class are used to create ObjInstance objects. When an ObjInstance object is created on the client, the corresponding Caché object is opened or created on the Caché server. ObjInstance objects contain the properties and methods of the Caché objects that they mirror.

5.2 The Factory Class

Instances of the Caché CacheActiveX.Factory class are used to create and manage connections to a Caché server. A Factory object also manages the creation of new ActiveX **ObjInstance** objects, either by opening an existing Caché object or by creating a new one.

The Factory class supports the following methods:

- Connect connects to Caché.
- ConnectDlg opens a connection dialog box and constructs a connection string from user input.
- Disconnect disconnects from Caché.
- DynamicSQL prepares a dynamic SQL statement and creates an object for execution.
- GetClassName extracts the class name from an OID.
- GetConnectionList returns the list of available connection strings from the client machine's registry.
- GetErrorText parses error messages
- GetId extracts the ID from an OID.
- GetLastErrorCount returns the number of errors in last status code.
- GetLastErrorNumber, returns the number of the last ObjectScript error.
- GetLastErrorParam, returns the value of a parameter for the last error.
- GetLastErrorParamCount, returns the number of parameters for the last error.
- IsConnected, reports if a connection has been established.
- IsMultibyte, indicates whether Unicode characters require special representation on the server.
- New, creates a new object on the server and a corresponding ActiveX proxy object on the client.
- Open, uses an OID to open an existing object on the server and a corresponding ActiveX proxy object on the client.
- OpenId, opens an existing object on the server and a corresponding ActiveX proxy object on the client.
- ResultSet, creates a Caché ActiveX ResultSet object that contains a previously defined query.
- SetOutput, provides callback functionality
- Static, executes the class method of an object on the Caché server through its corresponding client object.
- TransactionCommit, marks the end of a transaction.
- TransactionLevel, determines the current transaction level.
- TransactionRollback, terminates the current transaction and restores changes made by it.
- TransactionStart, marks the beginning of a transaction.

5.2.1 Connect()

Creates a connection to a specified namespace on a specified Caché server.

```
Boolean Connect(String connectstring)
```

See Defining a Connection String for full details on the `connectstring` format.

Parameter:

- `connectstring` — the connection string.

Returns:

a Boolean `true` if the connection attempt was successful, or `false` otherwise.

Example:

See Defining a Connection String.

5.2.2 ConnectDlg()

Provides access to the connection management dialog.

```
String ConnectDlg(String title)
```

This dialog allows users to add, edit, delete, or select a connection. When the user selects a connection, the dialog returns a complete and properly formatted connection string; if there is not enough information to create a complete connection string, the selection fails and the dialog remains open.

If the user completes the dialog, the method returns a connection string to pass to the **Connect()** method. If the user cancels the dialog, the method returns an empty string ("").

Parameter:

- `title` — Optional. Lets you supply a title for the connection dialog window. If *title* is not supplied, the default window name appears in the connection dialog window. Also, if “`+%up`” is appended to the *title* string, the method returns the username and password that the user has specified in the dialog

Returns:

a String

Example:

```
Set factory = CreateObject("CacheActiveX.Factory")

' set a custom title for dialog and specify that the dialog
' will return the username and password in the connection string
Dim MyTitle As String
MyTitle = "Application Connection Dialog+%up"

' get the connection string from user input to the connection dialog
sdir = factory.ConnectDlg(MyTitle)

' check if it's not NULL;
' if not NULL, use it to connect to the server
If sdir <> "" Then
    ok = factory.Connect(sdir)
    ' if the connection succeeds, put up a message box saying so
    MsgBox ok
Else
   ' if it's NULL, exit the program
   End
End If
```

5.2.3 Disconnect()

Closes a connection previously made by that object's **Connect()** method.

```
Boolean Disconnect()
```

This method always succeeds and returns `true`. This means that the application must perform any housekeeping to ensure that there are no open objects when the connection is broken.

Returns:

a Boolean `true` (never returns `false`).

Example:

```
' Given an open object Person and an open ResultSet object res,
' these must be closed before the connection is closed:
Person.SYS_Close
Set Person = Nothing


res.Close
Set res = Nothing

' Close the Factory object's connection:
factory.disconnect
```

Before you can close the connection, make sure that each object's reference count is zero — this prevents dangling references. To set an object's reference count to zero, you must break the association between the Caché client object and its corresponding server object. Once this is done for all objects, you may then shut down and deallocate memory for their client proxies.

5.2.4 DynamicSQL()

Creates a Caché ActiveX **ResultSet** object that contains a dynamically-defined query.

```
CacheActiveX.ResultSet DynamicSQL(String SQLStmt)
```

(There is no difference between the kind of ResultSet object created by this method and the kind created by the **ResultSet()** method of the Factory class.)

The query is applied to the database via the **Execute()** method of the returned ResultSet object. Afterward, you can retrieve data using the ResultSet object's various available methods. (If there are problems such as with the SQL statement, **DynamicSQL()** raises a Visual Basic error.)

Parameter:

- `SQLStmt` — An SQL statement that has been generated on the client at runtime.

Returns:

a ResultSet object.

Example:

```
' declare and create Factory object, declare the ResultSet object
Dim factory As CacheActiveX.factory
Set factory = CreateObject("CacheActiveX.Factory")

' Code here connects to the server; once the connection is
' established, work with the query can proceed.

Dim res As CacheActiveX.ResultSet

' declare variable to hold query string and put the query in it
Dim MyQuery As String
MyQuery = "SELECT %ID,Name,Colors,MyDate " _
        & "FROM Person " _
        & "WHERE (Name %STARTSWITH ?) " _
        & "ORDER BY Name"

' create the ResultSet using DynamicSQL()
Set res = factory.DynamicSQL(MyQuery)
```

```
' you can now set parameters, execute, and so on
Call res.SetParam(1, "F")
res.Execute
```

5.2.5 GetClassName()

Extracts the class name from a supplied OID.

```
String GetClassName(String OID)
```

Parameter:

- `OID` — OID of the desired class.

Returns:

a String

Example:

```
Dim MyClassName As String
MyClassName = factory.GetClassName(OID)
```

5.2.6 GetConnectionList()

Returns a list of connection strings.

```
String GetConnectionList()
```

The list is obtained from the client machine's registry in the HKEY_LOCAL_MACHINE_ISC_SERVERS key.

Note: The connection string for this method does not include the namespace (unlike those for other methods).

Returns:

a String containing a semi-colon-delimited list of connection names and strings in the following form:

```
ConnectionName1,Connection1;ConnectionName2,Connection2;...
```

Each connection is of the form:

```
cn_iptcp:ip_address[port]
```

Example:

```
Private m_factory As CacheActiveX.Factory

Set m_factory = CreateObject("CacheActiveX.Factory")
Debug.Print m_factory.GetConnectionList
```

The resulting string will look something like this:

```
LOCALTCP,cn_iptcp:127.0.0.1[1972]:;otherServer,cn_iptcp:123.123.123.123[1972]:
```

5.2.7 GetErrorText()

Takes the Caché object's error number and returns locale-dependent text for the generated error.

```
String GetErrorText(int ErrorNumber [, Param1, ..., Param10])
```

You can optionally pass it one or more parameters for inclusion in the error message. The return value appears in the language of the client's locale.

Parameters:

- `ErrorNumber` — a Caché error which was returned by the server.
- `ParamN` — each of the *ParamN* arguments supplies a value to be substituted into the text of the error message

Returns:

a String

Example:

```
' Caché was unable to save the class.
' The returned error is encapsulated by the error object,
'    #5659: "Property '%1' required"
msg = factory.GetErrorText(5659,"Name")
' msg now contains the string "Property 'Name' required"
```

5.2.8 GetId()

Returns the ID of an object given its OID.

```
String GetId(String OID)
```

Parameter:

- `OID` — a string that includes the class name and persistent ID of an object.

Returns:

a String

Example:

```
Dim ID As String
ID = factory.GetId(OID)
```

5.2.9 GetLastErrorCount()

Retrieves the number of errors in the last (most recently encountered) status code.

```
Integer GetLastErrorCount()
```

Note: The last error is per process, not per object.

Returns:

an Integer

Example:

```
Dim ErrCnt As Integer
ErrCnt = factory.GetLastErrorCount()
```

5.2.10 GetLastErrorNumber()

Retrieves the number of the last (most recently encountered) ObjectScript error.

```
Long GetLastErrorNumber(String ErrNumber)
```

Note: The last error is per process, not per object.

Parameter:

- `ErrNumber` — the optionally passed position of the error in the status code (1 by default).

Returns:

a Long integer between 1 and **GetLastErrorCount()** (inclusive).

Example:

```
Dim ErrNum As Long
ErrNum = factory.GetLastErrorNumber()
```

5.2.11 GetLastErrorParam()

Returns the value of the last (most recently encountered) error's parameter number.

```
String GetLastErrorParam([String ErrNumber[, String ParamNumber]])
```

Note: The last error is per process, not per object.

Parameters:

- `ErrNumber` — the optionally passed position of error in the status code (1 by default).
- `ParamNumber` — the optionally passed parameter number in this error (1 by default).

Returns:

a String

Example:

```
Dim ErrParam As String
ErrParam = factory.GetLastErrorParam()
```

or

```
Dim ErrParam As String
ErrParam = factory.GetLastErrorParam(errno)
```

or

```
Dim ErrParam As String
ErrParam = factory.GetLastErrorParam(errno, paramno)
```

Hence, the following calls are all equivalent:

```
Dim ErrParam As String
ErrParam = factory.GetLastErrorParam()
ErrParam = factory.GetLastErrorParam(1)
ErrParam = factory.GetLastErrorParam(1,1)
```

5.2.12 GetLastErrorParamCount()

Returns the number of parameters for the last (most recently encountered) error.

```
Integer GetLastErrorParamCount([String ErrNumber])
```

Note: The last error is per process, not per object.

Parameter:

- `ErrNumber` — the optionally passed position of the error in the status code (1 by default).

Returns:

an Integer

Example:

```
Dim ParamCount As Integer
ParamCount = factory.GetLastErrorParamCount()
```

or

```
Dim ParamCount As Integer
ParamCount = factory.GetLastErrorParamCount(errno)
```

5.2.13 IsConnected()

Returns `true` if a connection has been established.

```
Boolean IsConnected()
```

The return value of this method indicates whether or not the most recent connection operation succeeded or not.

Important: If a connection was established and was then broken by some other environmental effect, the function may still return `true`. This is because the last operation succeeded, even if the connection no longer exists. While the method performs some basic attempts to determine if the connection still exists, these may not succeed.

Returns:

a Boolean `true` if a connection has been established, or `false` otherwise.

Example:

```
If Not factory.IsConnected()
' Establish connection
End If
```

5.2.14 IsMultibyte()

Indicates whether Unicode characters require special representation on the server.

```
Boolean IsMultibyte()
```

A value of `true` means that a Unicode character is stored as an “escaped” sequence of ASCII characters. A `false` return value means that Unicode is the native representation for characters. Hence, the method returns `false` if the Caché server is a Unicode installation and `true` if not.

The purpose of this method is to ensure proper management of Unicode values in a SysList object.

Returns:

a Boolean `true` if Unicode characters require special representation on the server, or `false` otherwise.

Example:

```
Dim im as Boolean
im = factory.IsMultibyte()
```

5.2.15 New()

Creates a new object on the server and a corresponding ActiveX proxy object on the client.

```
Object New(String ClassName[, Variant init])
```

When successfully invoked, the **New()** method returns a new **ObjInstance** proxy object of the specified class, and creates a corresponding new Caché object on the Server. To be able to use the object, the Caché client application must cast the object to the type of the Caché object that was being opened. This makes the Caché functionality of the object available to the application.

Parameters:

- `ClassName` — the name of a valid class defined on the Caché Server
- `init` — an optional value used to initialize the new object.

Returns:

an **ObjInstance** object.

Example:

```
Dim obj as Object
Set obj = factory.New(ClassName)
```

5.2.16 Open()

Uses an object identifier (OID) to create a new **ObjInstance** proxy object populated with data from the object opened on the Caché server.

```
Object Open(String ClassName, String OID, [[Int concurrency], Int noreuse])
```

Note: This method is primarily for InterSystems internal use.

Parameters:

- `ClassName` — a class defined on the Caché server
- `OID` — an OID value.
- `concurrency` — Optional. Sets the level of concurrent usage allowed for this object.

- `noreuse` — Optional. Controls whether an object in memory can be referred to multiple times. This corresponds to the server-side ObjectScript syntax `##class(ClassName).%Open(oid)`.

Returns:

a new **ObjInstance** proxy object populated with data from the object opened on the Caché server.

Example:

```
Dim obj as Object
Set obj = factory.Open(ClassName,oid)
```

5.2.17 OpenId()

Creates a new **ObjInstance** proxy object populated with data from an object opened on the Caché server.

```
Object OpenId(String ClassName, String id, [[Int concurrency], Int noreuse])
```

Parameters:

- `ClassName` — a valid class defined on the Caché server
- `id` — an ID associated with a valid OID.
- `concurrency` — Optional. Sets the level of concurrent usage allowed for this object.
- `noreuse` — Optional. Controls whether an object in memory can be referred to multiple times.

Returns:

a new **ObjInstance** proxy object populated with data from an object opened on the Caché server.

Example:

```
Dim obj as Object
Set obj = factory.OpenId(ClassName,id)
```

5.2.18 ResultSet()

Creates a Caché ActiveX ResultSet object that contains a previously-defined query.

```
Object ResultSet(String ClassName, String QueryName)
```

There is no difference between the kind of ResultSet object created by this method and the kind created by the Factory class' DynamicSQL method.

The query is applied to the database via the **Execute()** method of the returned ResultSet object. Afterward, you can retrieve data using the ResultSet object's various available methods.

Parameters:

- `ClassName` — name of the class containing the query.
- `QueryName` — name of the class query to be used.

Returns:

a ResultSet object.

Example:

```
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")

Dim ok As Boolean
ok = res.Execute("L")  ' pass in any arguments expected by the query
while res.Next()
  Debug.Print res.Get("Name")
wend
```

5.2.19 SetOutput()

Allows you to direct server output (produced by a **Write** command in ObjectScript code) to a control that is part of a client application.

```
Sub SetOutput(Variant DisplayControl)
```

The control receiving the output from the server must have a text property where the output can appear.

This content must come from a single **Write** command. Each subsequent invocation of the **Write** will overwrite the content displayed by the last one. Similarly, if there are multiple pieces of input, they must be concatenated into a single unit of input and passed to the **Write** in that combined form. (You can also pass the output to a text object other than the one ultimately displaying it, concatenate it there, and then pass it along for display.)

Parameter:

- `DisplayControl` — the control that is to receive the output, or an empty string.

Example:

```
Set factory = CreateObject("CacheActiveX.Factory")
factory.SetOutput Text1
' System output now goes to textbox Text1
```

To disable output functionality you can call **SetOutput()** with an empty string:

```
factory.SetOutput ""
```

5.2.20 Static()

Returns a static ActiveX object which allows you to invoke class methods on the Caché server.

```
Object Static(ClassName As String)
```

There is no corresponding object opened on the server, so you cannot invoke instance or property methods from this ActiveX object.

Parameter:

- `ClassName` — class name of the object to be returned.

Returns:

a static ActiveX object.

Example:

```
' invoke a class method;
' put the content in a textbox called txtInfoBox
Dim MyObj As Object
Set MyObj = factory.Static("Cinema.Utils")

' the Cinema.Utils.SendEmail ClassMethod returns a string,
' which then appears in the textbox
txtInfoBox.Text = MyObj.SendEmail(Addr,Msg)
```

5.2.21 TransactionCommit()

Marks the successful end of a transaction initiated by the corresponding **TransactionStart()** method.

```
Sub TransactionCommit()
```

TransactionCommit() is exactly analogous to the ObjectScript TCommit command. For more information on transactions generally, see the TStart reference page.

TransactionCommit() decrements the value of the **$TLEVEL** special variable and terminates the transaction only if **$TLEVEL** goes to 0. That is to say, a transaction is committed when **TransactionCommit()** has been called as many times as **TransactionStart()**. Calling **TransactionCommit()** when **$TLEVEL** is 0 results in an ObjectScript <COMMAND> error.

Example:

```
factory.TransactionCommit()
```

5.2.22 TransactionLevel()

Returns the current nesting level for transaction processing.

```
Integer TransactionLevel()
```

The number of **TransactionStart()** calls issued before Caché encounters a **TransactionCommit()** or **TransactionRollBack()** determines the nesting level. Each **TransactionStart()** command increments the special variable $TLEVEL by 1. Each **TransactionCommit()** command decrements **$TLEVEL** by 1. A **TransactionRollBack()** command resets **$TLEVEL** to 0.

Returns:

an Integer

Example:

```
Dim lvl as Integer
set lvl=factory.TransactionLevel()
```

5.2.23 TransactionRollBack()

Terminates the current transaction and restores all journaled database values to the values they held at the start of the transaction.

```
Sub TransactionRollBack()
```

Example:

```
factory.TransactionRollBack()
```

5.2.24 TransactionStart()

Marks the beginning of a transaction.

```
Sub TransactionStart()
```

Following **TransactionStart()**, database operations are journaled to enable a subsequent **TransactionCommit()** or **TransactionRollBack()** command.

TransactionStart() is exactly analogous to the ObjectScript TStart command. For more information on transactions generally, see the TStart reference page.

The rollback occurs no matter what the value of the $TLEVEL special variable is. Following **TransactionRollBack()**, **$TLEVEL** is set to 0. Calling **TransactionRollBack()** when **$TLEVEL** is 0 has no effect.

TransactionRollBack() is exactly analogous to the ObjectScript TRollBack command. For more information on transactions generally, see the TStart reference page.

Example:

```
       factory.TransactionStart()
```

5.3 The ResultSet Class

Caché provides interfaces for performing both pre-defined or dynamically-defined queries. For access to this interface from an ActiveX client, use an ActiveX ResultSet object. A ResultSet object allows to you to set up a query, execute it, and parse its results.

As part of an ActiveX client, the ResultSet object exists independently; on the Caché server, a ResultSet is a class member — just as an object or a property is a class member. Each ActiveX ResultSet object is tied to a particular query — one defined either dynamically or in a specific Caché class.

The Caché ResultSet object supports the following members:

- Close, closes a ResultSet object after it has been executed.
- ContainsID, determines whether a query contains an ID.
- Execute, executes the query referenced by the ResultSet object.
- Get, uses the name of the column to return the data value for the specified column.
- GetColumnCount, returns the number of columns in the ResultSet object.
- GetColumnHeader, returns the description of the column in the specified position.
- GetColumnName, returns the name of a specified column.
- GetData, returns the data value for the specified column.
- GetDataAsString, returns the value of a specified column in string format.
- GetDataByName, uses the name of the column to return the data value for the specified column.
- GetParamCount, returns the number of possible parameters for the query.
- GetParamName, returns the name of a specified parameter.
- IsDataNull, determines if the specified column has been defined.
- Next, advances to the next row of data.

- Prepare, specifies a dynamic SQL query to use instead of a class query.
- SetParam, sets the value of input parameters required by the query.
- SetRunTimeMode, sets the runtime mode of the query.

Important: The default SQL runtime mode for CacheActiveX.dll is `ODBC`. If your application uses the older CacheObject.dll, the runtime mode would be `Logical` (see Upgrading from CacheObject.dll and the SetRunTimeMode() method).

5.3.1 Close()

Closes a result set after you have executed the query that created it.

```
Boolean Close()
```

This method does not free the result set's memory. If you wish to execute a statement multiple times, you must close the ResultSet object between pair of calls to the **Execute()** method, making the sequence of calls something like:

```
Prepare 'prior to the first Execute call
Execute
Close
Execute
Close
```

Returns:

a Boolean `true` if the result set was successfully closed, or `false` otherwise.

Example:

```
res.Close
' res no longer contains the data returned by the query.
' If the ResultSet object is no longer to be used,
' its memory can be freed with:
Set res = Nothing
```

5.3.2 ContainsID()

Determines whether the current query returns an ID and, if so, which column contains it.

```
Integer ContainsID()
```

The method returns the number of the ID column or 0 if the ResultSet does not contain an ID column; since column numbers start at 1, a returned non-zero value is that of the column number exactly. For information on how a pre-defined query holds information on its ID column, see the section About CONTAINID in *Using Caché Objects*.

Note: It is possible to manually specify a non-existent or incorrect column as the value of the column containing the ID and this method will retrieve that value.

Returns:

an Integer

Example:

```
' This puts a message up saying whether or not
' the query contains an ID field.
MsgBox "ContainsID = " & res.ContainsID
```

5.3.3 Execute()

Executes the query defined by the ResultSet object. It passes any arguments specified to the query.

```
Boolean Execute([param0, param1, ... , paramN])
```

Parameters:

- `param1,...,paramN` — up to 16 optional parameters passed to the query on the server in the order specified. These parameters are identified as `param0` (zero) through `param15`, inclusive

Returns:

a Boolean `true` if the query executed successfully, or `false` otherwise.

Example:

```
' First, declare and prepare the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")

' Now run the ByName query. This query returns all
' people whose names begin with the letter specified.
res.Execute("A")
```

5.3.4 Get()

Returns the value of the data stored in a column of the current row of the ResultSet, where that column is specified by its name.

```
Variant Get(String ColumnName)
```

While certain situations require the use of the **Get()** method, the **GetData()** method — which uses the column number rather than column name — is faster.

Note: This is the equivalent of the **GetDataByName()** method and supersedes it.

Parameter:

- `ColumnName` — the name of the column from the ResultSet.

Returns:

a Variant containing the value of the data.

Example:

```
' Declare, prepare, and run the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

While res.Next()
  ' places all the matching names in a listbox
  lstMyList.AddItem res.Get("Name")
Wend
```

5.3.5 GetColumnCount()

Returns the number of columns in the result set.

```
Integer GetColumnCount()
```

Returns:

an Integer indicating the number of columns in the result set.

Example:

```
' Declare, prepare, and run the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

' Get the number of columns in the result set.
For i = 1 To res.GetColumnCount
    ' For each column, display its name in a listbox.
    lstColumns.AddItem res.GetColumnName(i)
Next
```

5.3.6 GetColumnHeader()

Returns a description of the data contained in the specified column of a pre-defined query.

```
String GetColumnHeader(Integer ColumnNumber)
```

Note: **Not implemented in CacheActiveX**

This method is only implemented in the older CacheObject.dll (see "Upgrading from CacheObject.dll"), not in CacheActiveX.dll. CacheActiveX will always return the column (property) name, ignoring what has been defined in the ROWSPEC *optionalDescription* for the column. For example, if a class query has a ROWSPEC that defines a new column header like:

`"prop1:%String:MyHeader,prop2,prop3"`,

then CacheActiveX will return `"prop1"` instead of `"MyHeader"`.

If no description exists (and for dynamically defined queries, they never exist), **GetColumnHeader()** returns the name of the specified column. For information on setting up a pre-defined query's column headers, see the About ROWSPEC section of the queries chapter of *Using Caché Objects*.

Parameter:

- `ColumnNumber` — the numeric position of the data in the ResultSet. Column numbers start with 1.

Returns:

a String containing a description of the data.

Example:

Warning: this example only works when using the older CacheObject.dll (see note above).

```
' Declare, prepare, and run the query.
Dim res As CacheObject.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

' Get the number of columns in the result set.
For i = 1 To res.GetColumnCount
    ' For each column, display its name and header
    ' in a listbox as "Name - Header".
    lstColumns.AddItem res.GetColumnName(i) & " - " & res.GetColumnHeader(i)
Next
```

5.3.7 GetColumnName()

Returns an identifier associated with a specified column.

`String GetColumnName(Integer ColumnNumber)`

The behavior of this method depends on whether the ResultSet object was generated from a pre-defined query (using **Factory.ResultSet()**) or a dynamically generated query (using **Factory.DynamicSQL()**). For pre-defined queries, it returns the name of the property stored in the column; for dynamically defined queries, it returns *either* the name of the property stored in the column *or*, if defined, the alias of the property, as specified in an AS clause of the query's SELECT statement.

Parameter:

- `ColumnNumber` — the numeric position of the data in the ResultSet. Column numbers start with 1.

Returns:

a String

Example:

```
' Declare, prepare, and run the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

' Get the number of columns in the result set.
For i = 1 To res.GetColumnCount
    ' For each column, display its name in a listbox.
    lstColumns.AddItem res.GetColumnName(i)
Next
```

5.3.8 GetData()

Returns the value of the data stored in a column (specified by its number) from the current row of the ResultSet.

`Variant GetData(Integer ColumnNumber)`

Parameter:

- `ColumnNumber` — the numeric position of the data in the ResultSet. Column numbers start with 1.

Returns:

a Variant containing the value of the data.

Example:

```
' Declare, prepare, and run the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

' Retrieve the data from the specified column
While res.Next
    lstMyList.AddItem res.GetData(4)
Wend
```

5.3.9 GetDataAsString()

Returns the value of the data stored in a column (specified by name) from the current row of the ResultSet, converted to a string.

```
String GetDataAsString(Integer ColumnNumber)
```

It is similar to the **GetData()** method, except that it performs a type conversion.

Parameter:

- `ColumnNumber` — the numeric position of the data in the ResultSet. Column numbers start with 1.

Returns:

a String

Example:

```
' Declare, prepare, and run the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

' Retrieve the data from the specified column
While res.Next
    lstMyList.AddItem res.GetDataAsString(4)
Wend
```

5.3.10 GetDataByName()

Note: The **GetDataByName()** method has been superseded by the **Get()** method.

Returns the value of the data stored in a column of the current row of the ResultSet, where that column is specified by its name.

```
Variant GetDataByName(String ColumnName)
```

While certain situations require the use of the **GetDataByName()** (or **Get()**) methods, the **GetData()** method — which uses the column number rather than column name — is faster.

Parameter:

- `ColumnName` — name of the column.

Returns:

a Variant containing the value of the data.

Example:

```
' Declare, prepare, and run the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

' Retrieve the data from the specified column
While res.Next
    lstMyList.AddItem res.GetDataByName("Name")
Wend
```

5.3.11 GetParamCount()

Returns the maximum number of arguments a specified query expects to be passed.

```
Integer GetParamCount()
```

Returns:

an Integer

Example:

```
' Declare and prepare the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")

' Find out how many parameters the query uses and display it in a textbox.
txtParamCount.Text = res.GetParamCount
```

5.3.12 GetParamName()

Returns the name of the argument specified.

```
String GetParamName(Integer ParamNumber)
```

For example, if you want to know the name of the first argument, pass `1` as an argument to **GetParamName()**.

Parameter:

- `ParamNumber` — number of the parameter to be returned.

Returns:

a String

Example:

```
' Declare and prepare the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")

' Display each query parameter in a listbox.
For i = 1 To res.GetParamCount
    lstParams.AddItem res.GetParamName(i)
Next
```

5.3.13 IsDataNull()

Checks if a particular column in the current row of the ResultSet contains a SQL NULL value.

```
Boolean IsDataNull(Integer ColumnNumber)
```

Parameter:

- `ColumnNumber` — the numeric position of the data in the ResultSet, where column numbers start at 1.

Returns:

a Boolean `true` if:

- There is no data in the column. That is, the column is non-existent or the data in the column is the empty string (specified by ObjectScript code on the server such as `Set person.DOB=""`).
- The data in the column is `$c(0)` and its datatype is not String.

Example:

```
' Declare, prepare, and run the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

' Check if each row of the ResultSet has NULL value in column 4.
While res.Next
    lstMyList.AddItem res.IsDataNull(4)
Wend
```

5.3.14 Next()

Advances the ResultSet cursor to the next row of data.

```
Boolean Next()
```

Next() must be called to move to the first row of data before calling **Get()** or **GetData()**.

Returns:

a Boolean `true` if the move succeeds and the now-current row of data exists, or `false` if no more data exists.

Example:

```
' Declare, prepare, and run the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

' Iterate through the ResultSet.
While res.Next
    ' All data processing occurs here;
    ' see Get, all GetData..., and IsDataNull methods for examples.
Wend
```

5.3.15 Prepare()

Specifies a dynamic SQL statement (or statements) to use as the query, instead of specifying a class query.

This method returns a status, which should be checked before proceeding.

The queries can contain parameters represented by ? characters within the query. The values of any parameters are supplied via the **Execute()** method.

5.3.16 SetParam()

Sets the value of input parameters for the query.

```
Sub SetParam(Integer Index, Variant Value)
```

Parameters:

- `Index` — represents the position of the parameter in the list of parameters. Parameter index numbers start with 1.
- `Value` — the value to pass through to the query for that parameter

Example:

```
' Declare and prepare the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")

Call res.SetParam(1, "F")

' Run the query, since it already has the necessary parameter(s).
res.Execute
```

5.3.17 SetRunTimeMode()

Sets the SQL runtime mode for the query to be executed.

```
Sub SetRunTimeMode(Integer mode)
```

This method is typically used with **GetDataAsString()**, also of the ResultSet class, because this influences how the string conversion happens.

For more information on the various modes, see the section Data Formats and Translation Methods in *Using Caché Objects*.

Parameter:

- `mode` — an Integer that represents one of the available modes:
  - 0 for LOGICAL mode
  - 1 for ODBC mode (similar to the system datatypes' **LogicalToODBC()** method)
  - 2 for DISPLAY mode (similar to the system datatypes' **LogicalToDisplay()** method)

Example:

```
' Declare, prepare, and run the query.
Dim res As CacheActiveX.ResultSet
Set res = factory.ResultSet("Person","ByName")
res.Execute("A")

' Iterate through the ResultSet.
While res.Next
    res.SetRunTimeMode 0

    ' processing based on having data in logical mode
Wend
```

5.4 The SysList Class

You can use Caché SysList objects to parse and manipulate Caché lists ($List format) from within Visual Basic. You can also use them to create and manipulate lists in this format independent of the Caché server. See "Using Lists in Visual Basic" for an example that uses SysList.

Lists are ordered collections of information. Each list element is identified by its position (slot) in the list. You can set the value of a slot's data or insert data at a slot. If you set a new value for a slot, that value is stored in the list. If you set the value for an already-existing slot, the new data overwrites the previous data and the slot assignments are not modified. If you insert data at an already-existing slot, the new list item increments the slot number of all subsequent slots. (Inserting a new item in the second slot slides the data currently in the second slot to the third slot, the object currently in the third slot to the fourth slot, and so on.)

SysList objects are instances of the %List data type and allow you to manipulate individual items within the list; the list can hold individual items or items that are lists. In ObjectScript, the functions for manipulating $List objects are $LIST, $LISTBUILD, $LISTDATA, $LISTFIND, $LISTGET, and $LISTLENGTH.

Certain aspects of the SysList object depend on whether it is associated with an 8–bit or Unicode Caché server. For more information, see the sections below on the IsMultibyte property and the section of the Add property on Adding a Unicode Value.

The Caché SysList object supports the following properties:

- Count property, contains the number of slots in the list (whether the items are individual elements or themselves lists).
- IsMultibyte property, contains information specifying if the server is 8-bit or Unicode.
- Item property, contains an individual (non-list) item in the list.
- ItemList property, contains a list item in the list as a pointer to an object.

The Caché SysList object supports the following methods:

- Add, adds an item to the end of the list.
- Clear, empties the list.
- Get, returns a string containing the Caché $List object converted from SysList format.
- Remove, removes an element from the list and consolidates the remaining list items.
- Set, transforms a Caché $List object (serialized in a string) into a SysList object.

5.4.1 Count Property

Returns the length of the list (that is, the number of elements in it).

```
Count As Long
```

This includes:

- Elements containing single items.
- Empty elements. A list can contain an empty slot by being defined with ObjectScript server code such as

  ```
  Set MyList = $ListBuild("Value1",,"Value2")
  ```

  This creates a NULL value in the second element.
- Elements containing items that are themselves lists.

Reads:

the number of elements in the list.

Note: Because this property is read-only, trying to set its value generates an error.

Example:

```
Dim Cars as CacheActiveX.SysList
Set Cars = salesman.CarsInLot()

Dim count as Integer
count = Cars.Count
' count contains the number of cars in Cars.
```

5.4.2 IsMultibyte Property

Indicates whether Unicode characters require special representation on the server.

```
IsMultibyte As Boolean
```

Reads:

A value of `true` means that a Unicode character is stored as an "escaped" sequence of ASCII characters. A `false` return value means that Unicode is the native representation for characters.

The default value for this property is `false`. Typically, the application sets the value of this property using the value returned by the Factory object's IsMultibyte method, as in the example below.

Example:

```
myList.IsMultibyte = factory.IsMultibyte()
```

You may or may not know if an object is compatible with an 8-bit system, depending on how it is created. For example, in this first case, you *do* know what *MyObj* is, so you can properly work with it in your current environment:

```
Dim MyObj as CacheActiveX.SysList
Set MyObj = Obj.A
```

In this second case, you do *not* know what *MyObj* is and need to use the **IsMultibyte()** method to determine how to properly work with it in your current environment:

```
Dim MyObj as CacheActiveX.SysList
Set MyObj = Create obj("CacheActiveX.SysList")
```

5.4.3 Item Property

Allows you to view or set the value of each item in the list.

```
Property Item(Index As Integer)
```

Reads/Writes:

an Integer

The property can be used to unpack a list as follows:

```
' MyList is a SysList object with 2 items
Dim Item1 as String
Dim Item2 as String

Item1 = MyList.Item(1)
Item2 = MyList.Item(2)
```

It can also be used to create a new SysList object or to modify an existing one:

```
' Create a new list, Colors
Dim Colors as CacheActiveX.SysList

Set Colors = CreateObject("CacheActiveX.SysList")
Colors.Item(1) = "Red"
Colors.Item(2) = "Blue"
Colors.Item(3) = "Green"

' The first argument of the AddColors method is defined
' to have a %List data type
person.AddColors(Colors)
```

Note: Delphi users should use brackets rather than parentheses (`x.Item[1]` rather than `x.Item(1)`) to specify the item index.

5.4.4 ItemList Property

Allows you to view or set the value of any item in the list which is itself a SysList object.

```
Property ItemList(Index As Integer) As CacheActiveX.SysList
```

Note: There is no way to determine if a list item is itself a list simply by looking its value. You must have the information by some other mechanism.

Reads/Writes:

an Integer

Example:

You can use the ItemList property to create entries in a list:

```
' Create a new list, Books, containing the
' SysList objects Book1 and Book2
Dim Books as CacheActiveX.SysList

Set Books = CreateObject("CacheActiveX.SysList")
Set Books.ItemList(1)=Book1
Set Books.ItemList(2)=Book2
```

You can also use it to unpack SysList objects embedded inside another SysList object as follows:

```
'Using the Cars example above, the list contains 3 cars
Dim Car1 as CacheActiveX.SysList
Dim Car2 as CacheActiveX.SysList
Dim Car3 as CacheActiveX.SysList

Car1 = Cars.ItemList(1)
Car2 = Cars.ItemList(2)
Car3 = Cars.ItemList(3)
' Car1, Car2, and Car3 are all SysList objects which can
' themselves be unpacked using Item and ItemList.
```

5.4.5 Add()

Adds an item to the end of a list.

```
Sub Add(Item)
```

Parameter:

- `Item` — item to be added.

Example:

```
' Using the list Colors
Colors.Add("Yellow")

' Add Yellow to the end of the list Colors, making it the
' fourth element in the list.
Dim Color4 as String
Color4 = Colors.Item(4)
' Color4 is Yellow
```

Adding a Unicode Value

You can place Unicode values in a SysList object by calling

```
Syslist.Add(ChrW(9824))
```

If the resulting SysList is later sent to an 8-bit cache server, then accessing the resulting $List object will cause a <WIDE CHAR> error. To ensure that the server gets the object in correct format (regardless of whether Caché is using 8-bit or Unicode characters), use code along the lines of:

```
Set Factory = CreateObject(CacheActiveX.Factory)
Set List = CreateObject(CacheActiveX.Syslist)
Factory.Connect(my_connection)
List.IsMultibyte = Factory.IsMultibyte
```

5.4.6 Clear()

Purges all data from a SysList object and eliminates its structure, as well.

```
Sub Clear()
```

Example:

```
' The data in the Cars SysList object is no longer useful

Cars.Clear()
' Cars is now an empty SysList object.
```

5.4.7 Get()

Takes a Caché ActiveX SysList object, converts it into a Caché $List object, and returns that object as a String.

```
Function Get() As String
```

Returns:

a String

Example:

```
' Using the list Colors
Dim MyColors as String

MyColors = Colors.Get()
```

5.4.8 Remove()

Removes an item from the list. It then contracts the list so that the elements are sequentially numbered from 1.

```
Sub Remove(Index As Integer)
```

Parameter:

- `Index` — index number of the item to be removed.

Example:

```
' Using the list Colors with Red, Blue, Green, and Yellow

Colors.Remove(2)
' This call removes Blue from the list, makes Green the
' second item in the list and makes Yellow the third item
' in the list.

Dim Color1 as String
Dim Color2 as String
Dim Color3 as String

Color1 = Colors.Item(1)
Color2 = Colors.Item(2)
Color3 = Colors.Item(3)
' Color1 is Red, Color2 is Green, Color3 is Yellow
```

5.4.9 Set()

Takes a Caché $List object (considered as a string of characters, that is a String). It converts this into a Caché ActiveX SysList object.

```
Sub Set(CacheList)
```

Parameter:

- `CacheList` — a Caché `$List`.

Example:

```
Dim NewList as CacheActiveX.SysList

Set NewList = CreateObject("CacheActiveX.SysList")
NewList.Set(x)
```

5.5 BinaryStream Class

Caché BinaryStream objects are used to create and manipulate binary stream data such as pictures.

The Caché BinaryStream class supports the following property:

- Data property, holds the actual data.

The Caché BinaryStream class supports the following methods:

- Clear, removes the contents of the stream from permanent storage.
- DeleteAttribute, deletes a specific attribute of the stream.
- FileRead, reads data from a file on the client and imports it into the stream.
- FileWrite, gets the stream's data from the server and saves it to a file on the client.
- GetAttribute, returns the value of a named attribute associated with the character stream.
- GetPicture, returns IPictureDisp.
- IsDefinedAttribute, returns `true` if the named attribute exists for the stream, `false` otherwise.
- NextAttribute, retrieves the next attribute name in the sequence of attribute name-value pairs.
- Read, reads data from a stream object.

- Rewind, moves to the beginning of the stream.
- SetAttribute, assigns an attribute to the stream.
- SetPicture, converts a Picture object into a binary stream.
- Write, writes data to the stream.

5.5.1 Data Property

Contains all of the data associated with the stream object.

```
Variant Data
```

Note: Caché has no way of ensuring that the Data property is large enough to hold all the stream's data. The application itself must do this.

Example:

```
' Person has a binary stream attribute, Image.
string = Person.Image.Data
' string contains the binary data of the picture
```

5.5.2 Clear()

Removes the contents of the stream from permanent storage.

```
Sub Clear()
```

This removes the permanent stream storage and any temporary stream, and restores the stream to its initial state. It also removes all the attributes associated with this stream.

Example:

```
' The Person class has a binary stream attribute Image
Person.Image.Clear()
```

5.5.3 DeleteAttribute()

Deletes a specific attribute of the stream.

```
Boolean DeleteAttribute(String Name)
```

An attribute is an array of strings containing name-value pairs that is associated with the stream.

Parameter:

- `Name` — the name of the attribute to delete.

Returns:

a Boolean `true` if the attribute previously existed, or `false` otherwise.

Example:

```
Person.Image.DeleteAttribute("MyAttrib")
```

5.5.4 FileRead()

Reads data from a file on the client and imports it into the stream.

```
Boolean FileRead(String filename)
```

Once read, the data is held in memory until the stream is closed.

Parameter:

- `filename` — the full path to the data to import.

Returns:

a Boolean `true` if the read is successful, or `false` otherwise.

Example:

```
' The Person class has a binary stream attribute Image
Person.Image.FileRead("c:\MyPicture.jpg")
```

5.5.5 FileWrite()

Gets the stream's data from the server and saves it to a file on the client.

```
Boolean FileWrite(String filename)
```

Parameter:

- `filename` — the full path of the file for saving the data.

Returns:

a Boolean `true` if the write is successful, or `false` otherwise.

Example:

```
' The Person class has a binary stream attribute Image
Person.Image.FileWrite("c:\MyPicture.jpg")
```

5.5.6 GetAttribute()

Returns the value of a named attribute associated with the character stream.

```
String GetAttribute(String Name, String Default)
```

An attribute is an array of strings containing name-value pairs that is associated with the stream.

Parameters:

- `Name` — the name of the attribute.
- `Default` — the default value to be returned if the attribute has not been defined.

Returns:

a String

Example:

```
' The Person class has a binary stream attribute Image
Attr1Value=Person.Image.GetAttribute("Attr1")
```

or

```
' The Person class has a binary stream attribute Image
Attr1Value=Person.Image.GetAttribute("Attr1",1)
```

5.5.7 GetPicture()

Returns an IPictureDisp.

```
StdPicture GetPicture()
```

Note: Visual Basic 2005 stopped using the interface that Visual Basic 6 was using: Visual Basic 2005 uses Image for its picture control.

If you are using Visual Basic 6, the picture control in Visual Basic corresponds to what **GetPicture()** returns, so no additional work is needed. If you are using Visual Basic 2005, you will need to write a wrapper that will get the binary data and create an Image object from it. In order to do this, do not call **GetPicture()** at all. Instead, implement a .NET binary stream interface using the BinaryStream class.

Returns:

a StdPicture object (IPictureDisp).

5.5.8 IsDefinedAttribute()

Returns `true` if the named attribute exists for the stream.

```
Boolean IsDefinedAttribute(String Name)
```

Parameter:

- `Name` — the name of the attribute you want to test the existence of. An attribute is an array of strings containing name-value pairs that is associated with the stream.

Returns:

a Boolean `true` if the named attribute exists for the stream, or `false` otherwise.

Example:

```
'The Person class has a binary stream attribute Image
Dim exists as Boolean
exists=Person.Image.IsDefinedAttribute("MyAttrib")
```

5.5.9 NextAttribute()

Retrieves the next attribute name in the sequence of attribute name-value pairs.

```
String NextAttribute(String AttrMatch)
```

It operates on a string whose content is the name you want to start with. The method skips any names that begin with “%”. (An attribute is an array of strings containing name-value pairs that is associated with the stream.)

Parameter:

- `AttrMatch` — a string that occurs before the first name you want returned. Pass through an empty string ("") to start at the first attribute; otherwise pass through a string that occurs *before* the first name you want to find.

Returns:

a String

Example:

```
'The Person class has a binary stream attribute Image
Person.Image.NextAttribute("")
```

or

```
'The Person class has a binary stream attribute Image
Person.Image.NextAttribute("MyAttrib")
```

5.5.10 Read()

Returns a specified number of bytes from the Stream.

```
String Read(Int bytes)
```

Because Visual Basic uses Unicode characters, each character of a string is represented by even number of bytes. This means that calling **Read()** with an argument that specifies an odd number of bytes to be read results in half of a character being read from the last byte.

Read() reconciles this situation by returning a string consisting of only the whole characters it has read. It discards the last partial character. It does, however, properly position the file pointer so it points to the next unread byte.

For example, suppose the file consists of the Unicode characters for the string “Help”. In this case, `Read(5)` returns “He” and leaves the file position partway through the “l”. `Read(1)` returns an empty string and leaves the file location between the “l” and the “p”. “Read(2)” returns “p” and leaves the file positioned at the end.

If the actual BSTR returned by **BinaryStream.Read()** contains an odd number of bytes, the last byte cannot be accessed using regular Visual Basic string manipulation functions. To work with such a stream, either copy the string to a byte array or access it with functions such as **LenB**, **MidB**, and **AscB**. Additional information on this issue is available in Chapter 6 of *Win32 API Programming with Visual Basic*, which is online at the Microsoft Developer Network (MSDN) site.

Parameter:

- `chars` — the set of bytes to be read from the stream.

Returns:

a String containing a specified number of bytes (`chars`) from the Stream.

Example:

```
'The Person class has a binary stream attribute image
'Grab the first 100 bytes
Image1.Picture=Person.Image.Read(100)
```

5.5.11 Rewind()

Moves to the beginning of the stream.

```
Boolean Rewind()
```

Returns:

a Boolean `true` if the move is successful, or `false` otherwise.

Example:

```
' The Person class has a binary stream attribute Image
' Rewind it before using it
Person.Image.Rewind
```

5.5.12 SetAttribute()

Assigns an attribute to the stream.

```
Sub SetAttribute(String Name, Value)
```

An attribute is a name-value pair stored in an array associated with a stream.

Parameters:

- `Name` — the name of the attribute to set.
- `Value` — the value to be assigned to the attribute. Value can be of any primitive type (that is, it cannot be an array or an object reference) and must be quoted if it is a string.

Example:

```
'The Person class has a binary stream attribute Image
Person.Image.SetAttribute("ShoeSize",10)
```

5.5.13 SetPicture()

Converts a Visual Basic Picture object to a binary stream for the server.

```
Sub SetPicture(StdPicture picture)
```

Parameter:

- `picture` — the Visual Basic Picture to be converted.

Example:

```
' The Person class has a binary stream attribute Image
Set pic = photo.Image
Person.Image.SetPicture pic
```

5.5.14 Write()

Writes data to the stream.

```
Boolean Write(String data)
```

Parameter:

- `data` — the set of bytes to be written to the stream.

  Important: The argument must be a variable rather than string literal. If a literal is passed (such as `Person.Image.Write "test"`), a type mismatch error occurs.

Returns:

a Boolean `true` if the write is successful, or `false` otherwise.

Example:

```
' The Person class has a binary stream attribute Image
Person.Image.Write(data)
```

5.6 CharStream Class

Caché CharStream objects are used to create and manipulate character stream data such as journal entries and book chapters.

The Caché CharStream class supports the following property:

- Data property, contains all the data associated with a stream object.

The Caché CharStream class supports the following methods:

- Clear, removes the contents of the stream from permanent storage.
- DeleteAttribute, deletes a specific attribute of the stream.
- FileRead, reads data from a file on the client and imports it into the stream.
- FileWrite, gets the stream's data from the server and saves it to a file on the client.
- GetAttribute, returns the value of a named of an attribute associated with the stream.
- IsDefinedAttribute, returns `true` if the named attribute exists for the stream, `false` otherwise.
- NextAttribute, retrieves the next attribute name in the sequence of attribute name-value pairs.
- Read, returns a specified number of characters from the stream.
- Rewind, moves to the beginning of the stream's data.
- SetAttribute, assigns an attribute to the stream.
- Write, writes data to the stream.

5.6.1 Data Property

Contains all the data associated with the stream object.

```
Variant Data
```

Note: Caché has no way of ensuring that the Data property is large enough to hold all the stream's data. The application itself must do this.

Reads:

all the data associated with the stream object.

Example:

```
' Person has a character stream Memo
Text1.Text = Person.Memo.Data
' The textbox now displays the data in Memo
```

5.6.2 Clear()

Removes the contents of the stream from permanent and temporary storage.

```
Sub Clear()
```

This removes the permanent stream storage and any temporary stream, and restores the stream to its initial state. It also removes all the attributes associated with this stream.

Example:

```
' The Person class has a character stream Diary
Person.Diary.Clear()
```

5.6.3 DeleteAttribute()

Deletes a specific attribute of the stream.

```
Boolean DeleteAttribute(String AttrName)
```

An attribute is an array of strings containing name-value pairs that is associated with the stream.

Parameter:

- `AttrName` — the name of the attribute to delete.

Returns:

a Boolean `true` if the attribute previously existed, or `false` if it did not.

Example:

```
Person.Image.DeleteAttribute("MyAttrib")
```

5.6.4 FileRead()

Reads data from a file on the client and imports it into the stream.

```
Boolean FileRead(String filename)
```

Once read, the data is held in memory until the stream is closed.

Parameter:

- `filename` — the full path to the data to import.

Returns:

a Boolean `true` if the file data is successfully read and imported, or `false` otherwise.

Example:

```
' The Person class has a character stream Memo
Person.Memo.FileRead("c:\Memo.txt")
```

5.6.5 FileWrite()

Gets the stream's data from the server and saves it to a file on the client.

```
Boolean FileWrite(String filename)
```

Parameter:

- `filename` — the full path of the file for saving the data.

Returns:

a Boolean `true` if the write succeeds, or `false` otherwise.

Example:

```
' The Person class has a character stream Memo
Person.Memo.FileWrite("c:\Memo.txt")
```

5.6.6 GetAttribute()

Returns the value of a named of an attribute associated with the character stream.

```
String GetAttribute(String AttrName, String DefaultValue)
```

An attribute is an array of strings containing name-value pairs that is associated with the stream.

Parameters:

- `AttrName` — the name of the attribute.
- `DefaultValue` — the default value to be returned if the attribute has not been defined.

Returns:

a String

Example:

```
' The Person class has a character stream Image
Attr1Value=Person.Image.GetAttribute("Attr1")
```

or

```
' The Person class has a character stream Image
Attr1Value=Person.Image.GetAttribute("Attr1",1)
```

5.6.7 IsDefinedAttribute()

Returns `true` if the named attribute exists for the stream.

```
Boolean IsDefinedAttribute(String Name)
```

An attribute is an array of strings containing name-value pairs that is associated with the stream.

Parameter:

- `Name` — the name of the attribute you want to test the existence of.

Returns:

a Boolean `true` if the named attribute exists for the stream, or `false` otherwise.

Example:

```
'The Person class has a character stream Image

Dim exists as Boolean
exists=Person.Image.IsDefinedAttribute("MyAttrib")
```

5.6.8 NextAttribute()

Retrieves the next attribute name in the sequence of attribute name-value pairs.

```
String NextAttribute(String AttrMatch)
```

An attribute is an array of strings containing name-value pairs that is associated with the stream. The method operates on a string whose content is the name you want to start with. It skips any names that begin with `“%”`.

Parameter:

- `AttrMatch` — a string that occurs before the first attribute name you want returned. Pass through an empty string ("") to start at the first attribute's name; otherwise, pass through a string that occurs *before* the name value you want to find.

Returns:

a String containing the next attribute name in the sequence of attribute name-value pairs.

Example:

```
'The Person class has a character stream Image
Person.Image.NextAttribute("")
```

or

```
'The Person class has a character stream Image
Person.Image.NextAttribute("MyAttrib")
```

5.6.9 Read()

Returns a specified number of characters from the stream.

```
String Read(Int chars)
```

Parameter:

- `chars` — the number of characters to be read from the stream.

Returns:

a String

Example:

```
' The Person class has a character stream Memo
' Grab the first 10 characters
Text1.text=Person.Memo.Read(10)
```

5.6.10 Rewind()

Moves to the beginning of the stream.

```
Boolean Rewind()
```

Returns:

a Boolean `true` if the move is successful, or `false` otherwise.

Example:

```
' The Person class has a character stream Memo
' Rewind it before using it
Person.Memo.Rewind
```

5.6.11 SetAttribute()

Assigns an attribute to the stream.

```
Sub SetAttribute(String Name, Value)
```

An attribute is a name-value pair stored in an array associated with a stream.

Parameters:

- `Name` — the name of the attribute to set.
- `Value` — the value to be assigned to the attribute. Value can be of any primitive type (that is, it cannot be an array or an object reference) and must be quoted if it is a string.

Example:

```
'The Person class has a character stream Diary

Person.Image.SetAttribute("BGColor","ffffff")
```

5.6.12 Write()

Writes data to the stream.

```
Boolean Write(String data)
```

Parameter:

- `data` — a String containing the data to be written.

Returns:

a Boolean `true` if the write was successful, or `false` otherwise.

Example:

```
' The Person class has a character stream Memo
Person.Memo.Write(data)
```