



# Using ActiveX with Caché

Version 5.2  
01 September 2006

*Using ActiveX with Caché*

Caché Version 5.2 01 September 2006

Copyright © 2006 InterSystems Corporation.

All rights reserved.

This book was assembled and formatted in Adobe Page Description Format (PDF) using tools and information from the following sources: Sun Microsystems, RenderX, Inc., Adobe Systems, and the World Wide Web Consortium at [www.w3c.org](http://www.w3c.org). The primary document development tools were special-purpose XML-processing applications built by InterSystems using Caché and Java.



The Caché product and its logos are registered trademarks of InterSystems Corporation.



The Ensemble product and its logos are registered trademarks of InterSystems Corporation.



The InterSystems name and logo are trademarks of InterSystems Corporation.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

Caché, InterSystems Caché, Caché SQL, Caché ObjectScript, Caché Object, Ensemble, InterSystems Ensemble, Ensemble Object, and Ensemble Production are trademarks of InterSystems Corporation. All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

For Support questions about any InterSystems products, contact:

**InterSystems Worldwide Customer Support**

Tel: +1 617 621-0700

Fax: +1 617 374-9391

Email: [support@InterSystems.com](mailto:support@InterSystems.com)

# Table of Contents

<b>1 Introduction</b> .....	<b>1</b>
<b>2 The Elements of the ActiveX Interface</b> .....	<b>3</b>
2.1 The Caché Object Server for ActiveX .....	3
2.2 Caché ActiveX Objects .....	4
<b>3 Configuring a Visual Basic Project</b> .....	<b>7</b>
<b>4 The Parts of a Caché Objects/Visual Basic Application</b> .....	<b>9</b>
4.1 Connecting to a Server .....	9
4.2 Creating a New Object Instance .....	11
4.3 Saving an Object .....	11
4.4 Opening an Existing Object .....	12
4.5 Using Caché Objects in Visual Basic .....	13
4.6 Using Callback Functionality in Visual Basic .....	13
4.7 Running a Query in Visual Basic .....	14
4.8 Error Trapping in Visual Basic .....	14

# List of Figures

Caché Object Server for ActiveX .....	1
Automatic Continuation in Visual Basic Code .....	7
Pop-Up Help Window in Visual Basic .....	7

# List of Tables

ActiveX Error Codes ..... 15



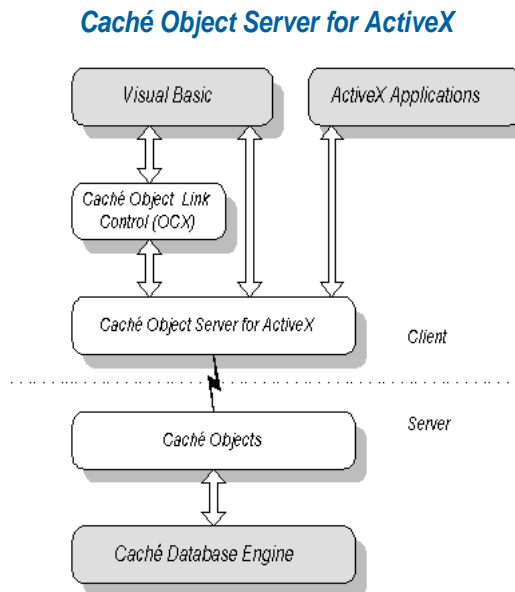
# 1

## 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 provides connectivity with a wide variety of client applications and development tools via an ActiveX interface:



Caché Objects includes 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 Visual Basic and other client applications, and provides a transparent interface to their server-side object instantiations. The binding also includes a special set of tools for use only from Visual Basic, which are collectively called Visual Caché.

**Note:** Although all of the examples in this document use Visual Basic, they do not use Visual Caché.

### 2.1 The Caché Object Server for ActiveX

The Caché Object Server for ActiveX 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 object that mirrors an object on a Caché server. Object properties are exposed as ActiveX properties. Similarly, methods are exposed as ActiveX methods. When methods are invoked, they are run on the server machine.

Every client-side Caché ObjInstance object refers to an 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. Refer to “[Caché ActiveX Objects](#)” below for more information on the types of objects available from ActiveX.

The client-side Caché object is an ActiveX object and is referred to using a pointer to the IDispatch interface (an “object” in Visual Basic). 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.

You can, if you like, generate a static ActiveX type library for your objects using the **\$system.OBJ.ExportODL(classlist,odlfile,flag,&errorlog)** routine, which is described in detail in the Caché Class Reference.

## 2.2 Caché ActiveX Objects

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

- **CacheObject.ObjInstance**—A client-side representation of a server-based Caché object. There is one instance of CacheObject.ObjInstance for each server-based Caché object that a client application uses.
- **CacheObject.Factory**—A factory object. Applications use this object to establish a connection to a Caché server and create and manage Caché objects. An application need only create one instance of the CacheObject.Factory class.
- **CacheObject.SysList**—A list manipulation object. Applications use this object to create and manipulate data in Caché’s [\\$List](#) format.
- **CacheObject.ResultSet**—A query processing object. Applications use this object 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.

- CacheObject.BinaryStream—A binary stream manipulation object. Applications use this object to create and manipulate data in Caché [binary streams](#).
- CacheObject.CharStream—A character stream manipulation object. Applications use this object to create and manipulate data in Caché [character streams](#).



# 3

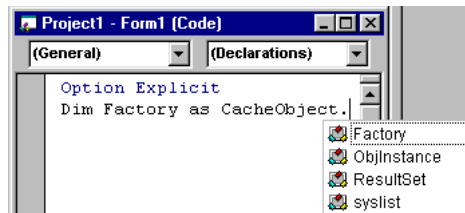
## Configuring a Visual Basic Project

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 CacheObject.SysList
```

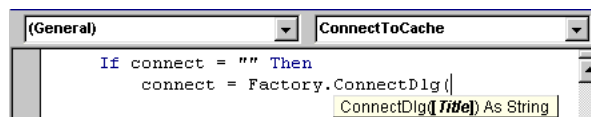
Early binding also provides pop up menus which show you possible elements to use to finish a statement. For example, you can choose from a list of possible object types to associate with a new instance:

### *Automatic Continuation in Visual Basic Code*



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

### *Pop-Up Help Window in Visual Basic*



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 cacheobject.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 CacheObject.

If CacheObject does not exist in a list of available references

1. Click on the **Browse** button.
2. Find cacheobject.dll in your <installation-root>\bin directory and double-click it.

# 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](#)
- [Error Trapping in Visual Basic](#)

### 4.1 Connecting to a Server

Typically, a client application first creates a `CacheObject.Factory` object and establish a connection to a server machine running Caché. Within Visual Basic the following code provides one way to do this:

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

Private Sub Form_Load()

' Create instance of factory object
' You must use "Set" to assign an object value
Set factory = CreateObject("CacheObject.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 `CacheObject.Factory` object by calling Visual Basic's **CreateObject** function:

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

`CreateObject` fails if it is unable to find or otherwise load `cacheobject.dll` (which contains the implementation of the `CacheObject.Factory` object).

As in the code above, you can optionally test a `CacheObject.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 `CacheObject.Factory` object, you then establish a connection to a server by using the **Connect** method of the `CacheObject.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 `CacheObject.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.

## 4.2 Creating a New Object Instance

You can create a new instance of a Caché object by using the CacheObject.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 this 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 CacheObject.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 `CacheObject.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 `CacheObject.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 Caché 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("CacheObject.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 CacheObject.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 Error Trapping in Visual Basic

The `CacheObject.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 CacheObject.Factory class. This method takes the description property of a generated *err* object and returns the text of the generated error. For example:

```

' Caché was unable to save your class MyClass.
' The returned error is encapsulated by the err object
msg = factory.GetErrorText(err.description)
' msg now contains the string
' "Unable to save class: MyClass"

```

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

### *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 <a href="#">Caché Error Reference</a> for the list of built-in messages.

