Early and Late Binding (Visual Basic)

Visual Studio 2015

The Visual Basic compiler performs a process called *binding* when an object is assigned to an object variable. An object is *early bound* when it is assigned to a variable declared to be of a specific object type. Early bound objects allow the compiler to allocate memory and perform other optimizations before an application executes. For example, the following code fragment declares a variable to be of type FileStream:

VB

```
'  Create a variable to hold a new object.
Dim FS As System.IO.FileStream
' Assign a new object to the variable.
FS = New System.IO.FileStream("C:\tmp.txt",
    System.IO.FileMode.Open)
```

Because FileStream is a specific object type, the instance assigned to FS is early bound.

By contrast, an object is *late bound* when it is assigned to a variable declared to be of type **Object**. Objects of this type can hold references to any object, but lack many of the advantages of early-bound objects. For example, the following code fragment declares an object variable to hold an object returned by the **CreateObject** function:

VB

```
' To use this example, you must have Microsoft Excel installed on your computer.
' Compile with Option Strict Off to allow late binding.
Sub TestLateBinding()
    Dim xlApp As Object
    Dim xlBook As Object
    Dim xlSheet As Object
    xlApp = CreateObject("Excel.Application")
    ' Late bind an instance of an Excel workbook.
    xlBook = xlApp.Workbooks.Add
    ' Late bind an instance of an Excel worksheet.
    xlSheet = xlBook.Worksheets(1)
    xlSheet.Activate()
    ' Show the application.
    xlSheet.Application.Visible = True
    ' Place some text in the second row of the sheet.
    xlSheet.Cells(2, 2) = "This is column B row 2"
End Sub
```

Advantages of Early Binding

You should use early-bound objects whenever possible, because they allow the compiler to make important

optimizations that yield more efficient applications. Early-bound objects are significantly faster than late-bound objects and make your code easier to read and maintain by stating exactly what kind of objects are being used. Another advantage to early binding is that it enables useful features such as automatic code completion and Dynamic Help because the Visual Studio integrated development environment (IDE) can determine exactly what type of object you are working with as you edit the code. Early binding reduces the number and severity of run-time errors because it allows the compiler to report errors when a program is compiled.

Note

Late binding can only be used to access type members that are declared as **Public**. Accessing members declared as **Friend** or **Protected Friend** results in a run-time error.

See Also

CreateObject
Object Lifetime: How Objects Are Created and Destroyed (Visual Basic)
Object Data Type

Determining Object Type (Visual Basic)

Visual Studio 2015

Generic object variables (that is, variables you declare as **Object**) can hold objects from any class. When using variables of type **Object**, you may need to take different actions based on the class of the object; for example, some objects might not support a particular property or method. Visual Basic provides two means of determining which type of object is stored in an object variable: the **TypeName** function and the **TypeOf...Is** operator.

TypeName and TypeOf...Is

The **TypeName** function returns a string and is the best choice when you need to store or display the class name of an object, as shown in the following code fragment:

VB

```
Dim Ctrl As Control = New TextBox
MsgBox(TypeName(Ctrl))
```

The **TypeOf...Is** operator is the best choice for testing an object's type, because it is much faster than an equivalent string comparison using **TypeName**. The following code fragment uses **TypeOf...Is** within an **If...Then...Else** statement:

VB

```
If TypeOf Ctrl Is Button Then
    MsgBox("The control is a button.")
End If
```

A word of caution is due here. The **TypeOf...Is** operator returns **True** if an object is of a specific type, or is derived from a specific type. Almost everything you do with Visual Basic involves objects, which include some elements not normally thought of as objects, such as strings and integers. These objects are derived from and inherit methods from Object. When passed an **Integer** and evaluated with **Object**, the **TypeOf...Is** operator returns **True**. The following example reports that the parameter `InParam` is both an **Object** and an **Integer**:

VB

```
Sub CheckType(ByVal InParam As Object)
    ' Both If statements evaluate to True when an
    ' Integer is passed to this procedure.
    If TypeOf InParam Is Object Then
        MsgBox("InParam is an Object")
    End If
    If TypeOf InParam Is Integer Then
        MsgBox("InParam is an Integer")
    End If
End Sub
```

The following example uses both **TypeOf...Is** and **TypeName** to determine the type of object passed to it in the `Ctrl` argument. The `TestObject` procedure calls `ShowType` with three different kinds of controls.

To run the example

1. Create a new Windows Application project and add a Button control, a CheckBox control, and a RadioButton control to the form.
2. From the button on your form, call the `TestObject` procedure.
3. Add the following code to your form:

VB

```
Sub ShowType(ByVal Ctrl As Object)
    'Use the TypeName function to display the class name as text.
    MsgBox(TypeName(Ctrl))
    'Use the TypeOf function to determine the object's type.
    If TypeOf Ctrl Is Button Then
        MsgBox("The control is a button.")
    ElseIf TypeOf Ctrl Is CheckBox Then
        MsgBox("The control is a check box.")
    Else
        MsgBox("The object is some other type of control.")
    End If
End Sub

Protected Sub TestObject()
    'Test the ShowType procedure with three kinds of objects.
    ShowType(Me.Button1)
    ShowType(Me.CheckBox1)
    ShowType(Me.RadioButton1)
End Sub
```

See Also

TypeName
Calling a Property or Method Using a String Name (Visual Basic)
Object Data Type
If...Then...Else Statement (Visual Basic)
String Data Type (Visual Basic)
Integer Data Type (Visual Basic)

Calling a Property or Method Using a String Name (Visual Basic)

Visual Studio 2015

In most cases, you can discover the properties and methods of an object at design time, and write code to handle them. However, in some cases you may not know about an object's properties and methods in advance, or you may just want the flexibility of enabling an end user to specify properties or execute methods at run time.

CallByName Function

Consider, for example, a client application that evaluates expressions entered by the user by passing an operator to a COM component. Suppose you are constantly adding new functions to the component that require new operators. When you use standard object access techniques, you must recompile and redistribute the client application before it could use the new operators. To avoid this, you can use the **CallByName** function to pass the new operators as strings, without changing the application.

The **CallByName** function lets you use a string to specify a property or method at run time. The signature for the **CallByName** function looks like this:

Result = **CallByName**(*Object*, *ProcedureName*, *CallType*, *Arguments*())

The first argument, *Object*, takes the name of the object you want to act upon. The *ProcedureName* argument takes a string that contains the name of the method or property procedure to be invoked. The *CallType* argument takes a constant that represents the type of procedure to invoke: a method (`Microsoft.VisualBasic.CallType.Method`), a property read (`Microsoft.VisualBasic.CallType.Get`), or a property set (`Microsoft.VisualBasic.CallType.Set`). The *Arguments* argument, which is optional, takes an array of type **Object** that contains any arguments to the procedure.

You can use **CallByName** with classes in your current solution, but it is most often used to access COM objects or objects from .NET Framework assemblies.

Suppose you add a reference to an assembly that contains a class named `MathClass`, which has a new function named `SquareRoot`, as shown in the following code:

VB

```
Class MathClass
    Function SquareRoot(ByVal X As Double) As Double
        Return Math.Sqrt(X)
    End Function
    Function InverseSine(ByVal X As Double) As Double
        Return Math.Atan(X / Math.Sqrt(-X * X + 1))
    End Function
    Function Acos(ByVal X As Double) As Double
        Return Math.Atan(-X / Math.Sqrt(-X * X + 1)) + 2 * Math.Atan(1)
    End Function
```

```
End Class
```

Your application could use text box controls to control which method will be called and its arguments. For example, if `TextBox1` contains the expression to be evaluated, and `TextBox2` is used to enter the name of the function, you can use the following code to invoke the `SquareRoot` function on the expression in `TextBox1`:

VB

```
Private Sub CallMath()
    Dim Math As New MathClass
    Me.TextBox1.Text = CStr(CallByName(Math, Me.TextBox2.Text,
       Microsoft.VisualBasic.CallType.Method, TextBox1.Text))
End Sub
```

If you enter "64" in `TextBox1`, "SquareRoot" in `TextBox2`, and then call the `CallMath` procedure, the square root of the number in `TextBox1` is evaluated. The code in the example invokes the `SquareRoot` function (which takes a string that contains the expression to be evaluated as a required argument) and returns "8" in `TextBox1` (the square root of 64). Of course, if the user enters an invalid string in `TextBox2`, if the string contains the name of a property instead of a method, or if the method had an additional required argument, a run-time error occurs. You have to add robust error-handling code when you use **CallByName** to anticipate these or any other errors.

Note

While the **CallByName** function may be useful in some cases, you must weigh its usefulness against the performance implications — using **CallByName** to invoke a procedure is slightly slower than a late-bound call. If you are invoking a function that is called repeatedly, such as inside a loop, **CallByName** can have a severe effect on performance.

See Also

CallByName
Determining Object Type (Visual Basic)

Working with Dynamic Objects (Visual Basic)

Visual Studio 2015

Dynamic objects provide another way, other than the **Object** type, to late bind to an object at run time. A dynamic object exposes members such as properties and methods at run time by using dynamic interfaces that are defined in the System.Dynamic namespace. You can use the classes in the System.Dynamic namespace to create objects that work with data structures that do not match a static type or format. You can also use the dynamic objects that are defined in dynamic languages such as IronPython and IronRuby. For examples that show how to create dynamic objects or use a dynamic object defined in a dynamic language, see Walkthrough: Creating and Using Dynamic Objects (C# and Visual Basic), DynamicObject, or ExpandoObject.

Visual Basic binds to objects from the dynamic language runtime and dynamic languages such as IronPython and IronRuby by using the IDynamicMetaObjectProvider interface. Examples of classes that implement the **IDynamicMetaObjectProvider** interface are the DynamicObject and ExpandoObject classes.

If a late-bound call is made to an object that implements the **IDynamicMetaObjectProvider** interface, Visual Basic binds to the dynamic object by using that interface. If a late-bound call is made to an object that does not implement the **IDynamicMetaObjectProvider** interface, or if the call to the **IDynamicMetaObjectProvider** interface fails, Visual Basic binds to the object by using the late-binding capabilities of the Visual Basic runtime.

See Also

- DynamicObject
- ExpandoObject
- Walkthrough: Creating and Using Dynamic Objects (C# and Visual Basic)
- Early and Late Binding (Visual Basic)

Object Lifetime: How Objects Are Created and Destroyed (Visual Basic)

Visual Studio 2015

An instance of a class, an object, is created by using the **New** keyword. Initialization tasks often must be performed on new objects before they are used. Common initialization tasks include opening files, connecting to databases, and reading values of registry keys. Visual Basic controls the initialization of new objects using procedures called *constructors* (special methods that allow control over initialization).

After an object leaves scope, it is released by the common language runtime (CLR). Visual Basic controls the release of system resources using procedures called *destructors*. Together, constructors and destructors support the creation of robust and predictable class libraries.

Using Constructors and Destructors

Constructors and destructors control the creation and destruction of objects. The **Sub New** and **Sub Finalize** procedures in Visual Basic initialize and destroy objects; they replace the **Class_Initialize** and **Class_Terminate** methods used in Visual Basic 6.0 and earlier versions.

Sub New

The **Sub New** constructor can run only once when a class is created. It cannot be called explicitly anywhere other than in the first line of code of another constructor from either the same class or from a derived class. Furthermore, the code in the **Sub New** method always runs before any other code in a class. Visual Basic 2005 and later versions implicitly create a **Sub New** constructor at run time if you do not explicitly define a **Sub New** procedure for a class.

To create a constructor for a class, create a procedure named **Sub New** anywhere in the class definition. To create a parameterized constructor, specify the names and data types of arguments to **Sub New** just as you would specify arguments for any other procedure, as in the following code:

VB

```
Sub New(ByVal s As String)
```

Constructors are frequently overloaded, as in the following code:

VB

```
Sub New(ByVal s As String, i As Integer)
```

When you define a class derived from another class, the first line of a constructor must be a call to the constructor of the base class, unless the base class has an accessible constructor that takes no parameters. A call to the base class that contains the above constructor, for example, would be `MyBase.New(s)`. Otherwise, `MyBase.New` is optional, and the Visual Basic runtime calls it implicitly.

After you write the code to call the parent object's constructor, you can add any additional initialization code to the **Sub New** procedure. **Sub New** can accept arguments when called as a parameterized constructor. These parameters are passed from the procedure calling the constructor, for example, `Dim AnObject As New ThisClass(X)`.

Sub Finalize

Before releasing objects, the CLR automatically calls the **Finalize** method for objects that define a **Sub Finalize** procedure. The **Finalize** method can contain code that needs to execute just before an object is destroyed, such as code for closing files and saving state information. There is a slight performance penalty for executing **Sub Finalize**, so you should define a **Sub Finalize** method only when you need to release objects explicitly.

> **Note**
>
> The garbage collector in the CLR does not (and cannot) dispose of *unmanaged objects*, objects that the operating system executes directly, outside the CLR environment. This is because different unmanaged objects must be disposed of in different ways. That information is not directly associated with the unmanaged object; it must be found in the documentation for the object. A class that uses unmanaged objects must dispose of them in its **Finalize** method.

The **Finalize** destructor is a protected method that can be called only from the class it belongs to, or from derived classes. The system calls **Finalize** automatically when an object is destroyed, so you should not explicitly call **Finalize** from outside of a derived class's **Finalize** implementation.

Unlike **Class_Terminate**, which executes as soon as an object is set to nothing, there is usually a delay between when an object loses scope and when Visual Basic calls the **Finalize** destructor. Visual Basic 2005 and later versions allow for a second kind of destructor, Dispose, which can be explicitly called at any time to immediately release resources.

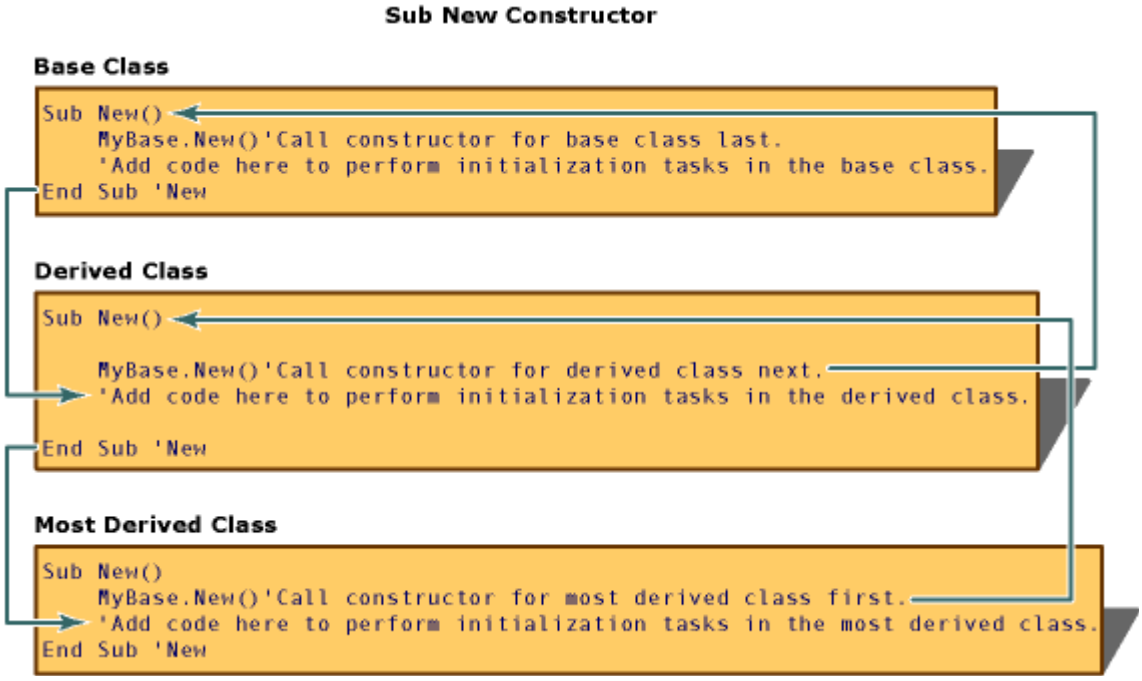
> **Note**
>
> A **Finalize** destructor should not throw exceptions, because they cannot be handled by the application and can cause the application to terminate.

How New and Finalize Methods Work in a Class Hierarchy

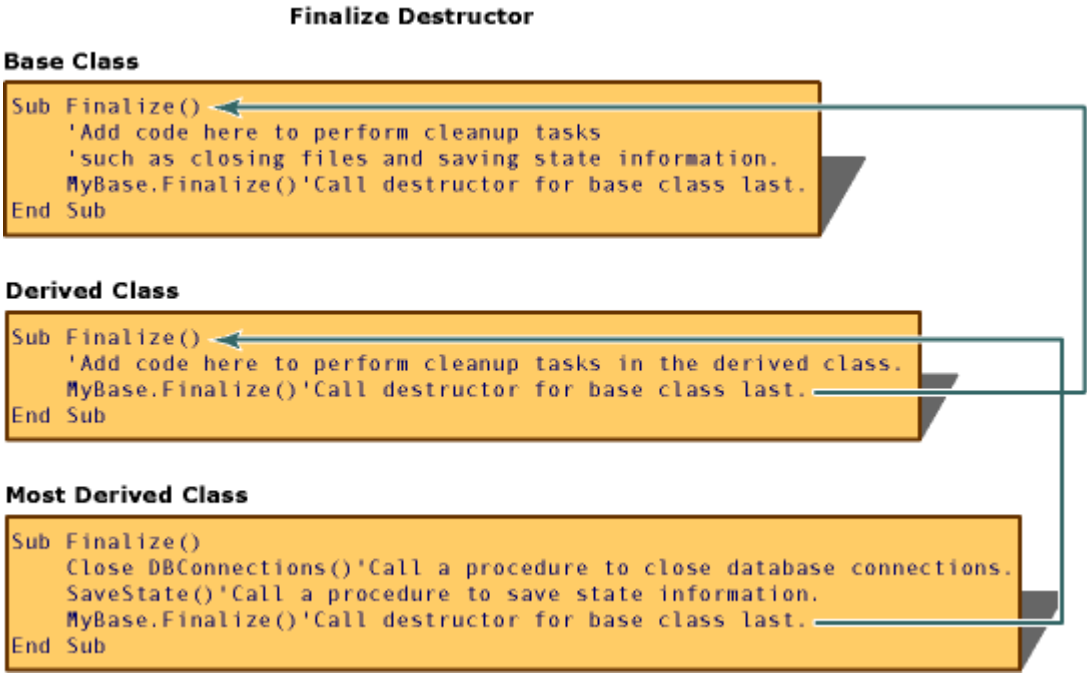
Whenever an instance of a class is created, the common language runtime (CLR) attempts to execute a procedure named **New**, if it exists in that object. **New** is a type of procedure called a *constructor* that is used to initialize new objects before any other code in an object executes. A **New** constructor can be used to open files, connect to databases, initialize variables, and take care of any other tasks that need to be done before an object can be used.

When an instance of a derived class is created, the **Sub New** constructor of the base class executes first, followed by constructors in derived classes. This happens because the first line of code in a **Sub New** constructor uses the syntax `MyBase.New()`to call the constructor of the class immediately above itself in the class hierarchy. The **Sub New** constructor is then called for each class in the class hierarchy until the constructor for the base class is reached. At that point, the code in the constructor for the base class executes, followed by the code in each constructor in all derived

classes and the code in the most derived classes is executed last.



When an object is no longer needed, the CLR calls the Finalize method for that object before freeing its memory. The Finalize method is called a *destructor* because it performs cleanup tasks, such as saving state information, closing files and connections to databases, and other tasks that must be done before releasing the object.



IDisposable Interface

Class instances often control resources not managed by the CLR, such as Windows handles and database connections. These resources must be disposed of in the **Finalize** method of the class, so that they will be released when the object is destroyed by the garbage collector. However, the garbage collector destroys objects only when the CLR requires more

free memory. This means that the resources may not be released until long after the object goes out of scope.

To supplement garbage collection, your classes can provide a mechanism to actively manage system resources if they implement the IDisposable interface. IDisposable has one method, Dispose, which clients should call when they finish using an object. You can use the Dispose method to immediately release resources and perform tasks such as closing files and database connections. Unlike the **Finalize** destructor, the Dispose method is not called automatically. Clients of a class must explicitly call Dispose when you want to immediately release resources.

Implementing IDisposable

A class that implements the IDisposable interface should include these sections of code:

- A field for keeping track of whether the object has been disposed:

```
Protected disposed As Boolean = False
```

- An overload of the Dispose that frees the class's resources. This method should be called by the Dispose and **Finalize** methods of the base class:

```
Protected Overridable Sub Dispose(ByVal disposing As Boolean)
    If Not Me.disposed Then
        If disposing Then
            ' Insert code to free managed resources.
        End If
        ' Insert code to free unmanaged resources.
    End If
    Me.disposed = True
End Sub
```

- An implementation of Dispose that contains only the following code:

```
Public Sub Dispose() Implements IDisposable.Dispose
    Dispose(True)
    GC.SuppressFinalize(Me)
End Sub
```

- An override of the **Finalize** method that contains only the following code:

```
Protected Overrides Sub Finalize()
    Dispose(False)
    MyBase.Finalize()
End Sub
```

Deriving from a Class that Implements IDisposable

A class that derives from a base class that implements the IDisposable interface does not need to override any of the base methods unless it uses additional resources that need to be disposed. In that situation, the derived class should override the base class's `Dispose(disposing)` method to dispose of the derived class's resources. This override must call the base class's `Dispose(disposing)` method.

```
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If Not Me.disposed Then
        If disposing Then
            ' Insert code to free managed resources.
        End If
        ' Insert code to free unmanaged resources.
    End If
    MyBase.Dispose(disposing)
End Sub
```

A derived class should not override the base class's Dispose and **Finalize** methods. When those methods are called from an instance of the derived class, the base class's implementation of those methods call the derived class's override of the `Dispose(disposing)` method.

Garbage Collection and the Finalize Destructor

The .NET Framework uses the *reference-tracing garbage collection* system to periodically release unused resources. Visual Basic 6.0 and earlier versions used a different system called *reference counting* to manage resources. Although both systems perform the same function automatically, there are a few important differences.

The CLR periodically destroys objects when the system determines that such objects are no longer needed. Objects are released more quickly when system resources are in short supply, and less frequently otherwise. The delay between when an object loses scope and when the CLR releases it means that, unlike with objects in Visual Basic 6.0 and earlier versions, you cannot determine exactly when the object will be destroyed. In such a situation, objects are said to have *non-deterministic lifetime*. In most cases, non-deterministic lifetime does not change how you write applications, as long as you remember that the **Finalize** destructor may not immediately execute when an object loses scope.

Another difference between the garbage-collection systems involves the use of **Nothing**. To take advantage of reference counting in Visual Basic 6.0 and earlier versions, programmers sometimes assigned **Nothing** to object variables to release the references those variables held. If the variable held the last reference to the object, the object's resources were released immediately. In later versions of Visual Basic, while there may be cases in which this procedure is still valuable, performing it never causes the referenced object to release its resources immediately. To release resources immediately, use the object's Dispose method, if available. The only time you should set a variable to **Nothing** is when its lifetime is long relative to the time the garbage collector takes to detect orphaned objects.

See Also

Dispose
Initialization and Termination of Components
New Operator (Visual Basic)
Cleaning Up Unmanaged Resources
Nothing (Visual Basic)

Interaction.CreateObject Method (String, String)

.NET Framework (current version)

Creates and returns a reference to a COM object. **CreateObject** cannot be used to create instances of classes in Visual Basic unless those classes are explicitly exposed as COM components.

Namespace: Microsoft.VisualBasic
Assembly: Microsoft.VisualBasic (in Microsoft.VisualBasic.dll)

Syntax

VB

```
<HostProtectionAttribute(SecurityAction.LinkDemand, Resources :=
HostProtectionResource.ExternalProcessMgmt)>
<SecurityPermissionAttribute(SecurityAction.Demand, Flags :=
SecurityPermissionFlag.UnmanagedCode)>
Public Shared Function CreateObject (
        ProgId As String,
        ServerName As String
) As Object
```

Parameters

ProgId
Type: System.String
Required. **String**. The program ID of the object to create.

ServerName
Type: System.String
Optional. **String**. The name of the network server where the object will be created. If *ServerName* is an empty string (""), the local computer is used.

Return Value

Type: System.Object
Creates and returns a reference to a COM object. **CreateObject** cannot be used to create instances of classes in Visual Basic unless those classes are explicitly exposed as COM components.

Exceptions

Exception	Condition
Exception	*ProgId* not found or not supplied -or- *ServerName* fails the **DnsValidateName** function, most likely because it is longer than 63 characters or contains an invalid character.
Exception	Server is unavailable
FileNotFoundException	No object of the specified type exists

Remarks

To create an instance of a COM component, assign the object returned by **CreateObject** to an object variable:

VB

```
Sub CreateADODB()
   Dim adoApp As Object
   adoApp = CreateObject("ADODB.Connection")
End Sub
```

The type of object variable you use to store the returned object can affect your application's performance. Declaring an object variable with the **As Object** clause creates a variable that can contain a reference to any type of object. However, access to the object through that variable is *late-bound*, that is, the binding occurs when your program runs. There are many reasons you should avoid late binding, including slower application performance.

You can create an object variable that results in early binding—that is, binding when the program is compiled. To do so, add a reference to the type library for your object from the **COM** tab of the **Add Reference** dialog box on the **Project** menu. Then declare the object variable of the specific type of your object. In most cases, it is more efficient to use the **Dim** statement and a primary interop assembly to create objects than it is to use the **CreateObject** function.

Interacting with Unmanaged Code

Another issue is that COM objects use unmanaged code — code without the benefit of the common language runtime. There is a fair degree of complexity involved in mixing the managed code of Visual Basic with unmanaged code from COM. When you add a reference to a COM object, Visual Basic searches for a primary interop assembly (PIA) for that library; if it finds one, then it uses it. If it does not find a PIA, then it creates an interoperability assembly that contains local interoperability classes for each class in the COM library. For more information, see COM Interoperability in .NET Framework Applications (Visual Basic).

You should generally use strongly bound objects and primary interop assemblies whenever possible. The examples below use the **CreateObject** function with Microsoft Office objects for demonstration purposes only. However, these objects are easier to use and more reliable when used with the appropriate primary interop assembly.

Creating an Object on a Remote Computer

You can create an object on a remote networked computer by passing the name of the computer to the *ServerName* argument of the **CreateObject** function. That name is the same as the Machine Name portion of a share name: for a share named "\\MyServer\Public," *ServerName* is "MyServer."

Note

Refer to COM documentation (see Microsoft Developer Network) for additional information on making an application accessible on a remote networked computer. You may need to add a registry key for your application.

The following code returns the version number of an instance of Excel running on a remote computer named `MyServer`:

VB

```
Sub CreateRemoteExcelObj()
    Dim xlApp As Object
    ' Replace string "\\MyServer" with name of the remote computer.
    xlApp = CreateObject("Excel.Application", "\\MyServer")
    MsgBox(xlApp.Version)
End Sub
```

If the remote server name is incorrect, or if it is unavailable, a run-time error occurs.

Note

Use **CreateObject** when there is no current instance of the object. If an instance of the object is already running, a new instance is started, and an object of the specified type is created. To use the current instance, or to start the application and have it load a file, use the **GetObject** function. If an object has registered itself as a single-instance object, only one instance of the object is created, no matter how many times **CreateObject** is executed.

Creating Framework Objects

You can use the **CreateObject** function only to create a COM object. While there is no exact equivalent mechanism for creating a .NET Framework object, the Activator in the System namespace contains methods to create local or remote objects. In particular, the CreateInstance method or the CreateInstanceFrom method might be useful.

Security Note

The **CreateObject** function requires unmanaged code permission, which might affect its execution in partial-trust situations. For more information, see SecurityPermission and Code Access Permissions.

Examples

The following example uses the **CreateObject** function to create a Microsoft Excel worksheet and saves the worksheet to a file. To use this example, Excel must be installed on the computer where this program runs. Also, you must add a reference to the type library from the **COM** tab of the **Add Reference** dialog box on the **Project** menu. The name of the type library varies depending on the version of Excel installed on your computer. For example, the type library for Microsoft Excel 2002 is named **Microsoft Excel 10.0 Object Library**.

VB

```
Sub TestExcel()
    Dim xlApp As Microsoft.Office.Interop.Excel.Application
    Dim xlBook As Microsoft.Office.Interop.Excel.Workbook
    Dim xlSheet As Microsoft.Office.Interop.Excel.Worksheet

    xlApp = CType(CreateObject("Excel.Application"),
                  Microsoft.Office.Interop.Excel.Application)
    xlBook = CType(xlApp.Workbooks.Add,
                  Microsoft.Office.Interop.Excel.Workbook)
    xlSheet = CType(xlBook.Worksheets(1),
                  Microsoft.Office.Interop.Excel.Worksheet)

    ' The following statement puts text in the second row of the sheet.
    xlSheet.Cells(2, 2) = "This is column B row 2"
    ' The following statement shows the sheet.
    xlSheet.Application.Visible = True
    ' The following statement saves the sheet to the C:\Test.xls directory.
    xlSheet.SaveAs("C:\Test.xls")
    ' Optionally, you can call xlApp.Quit to close the workbook.
End Sub
```

Version Information

.NET Framework
Available since 1.1

See Also

- GetObject
- Exception
- FileNotFoundException
- Activator
- CreateInstance
- CreateInstanceFrom
- Interaction Class
- Microsoft.VisualBasic Namespace
- Dim Statement (Visual Basic)
- Declare Statement
- COM Interoperability in .NET Framework Applications (Visual Basic)
- Interoperating with Unmanaged Code

Return to top

DynamicObject Class

.NET Framework (current version)

Provides a base class for specifying dynamic behavior at run time. This class must be inherited from; you cannot instantiate it directly.

Namespace: System.Dynamic
Assembly: System.Core (in System.Core.dll)

Inheritance Hierarchy

System.Object
  System.Dynamic.DynamicObject
    System.Windows.Interop.DynamicScriptObject

Syntax

VB

```
<SerializableAttribute>
Public Class DynamicObject
        Implements IDynamicMetaObjectProvider
```

Constructors

	Name	Description
	DynamicObject()	Enables derived types to initialize a new instance of the DynamicObject type.

Methods

	Name	Description
	Equals(Object)	Determines whether the specified object is equal to the current object.(Inherited from Object.)

	Finalize()	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection.(Inherited from Object.)
	GetDynamicMemberNames()	Returns the enumeration of all dynamic member names.
	GetHashCode()	Serves as the default hash function. (Inherited from Object.)
	GetMetaObject(Expression)	Provides a DynamicMetaObject that dispatches to the dynamic virtual methods. The object can be encapsulated inside another DynamicMetaObject to provide custom behavior for individual actions. This method supports the Dynamic Language Runtime infrastructure for language implementers and it is not intended to be used directly from your code.
	GetType()	Gets the Type of the current instance.(Inherited from Object.)
	MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object.)
	ToString()	Returns a string that represents the current object. (Inherited from Object.)
	TryBinaryOperation(BinaryOperationBinder, Object, Object)	Provides implementation for binary operations. Classes derived from the DynamicObject class can override this method to specify dynamic behavior for operations such as addition and multiplication.
	TryConvert(ConvertBinder, Object)	Provides implementation for type conversion operations. Classes derived from the DynamicObject class can override this method to specify dynamic behavior for operations that convert an object from one type to another.
	TryCreateInstance(CreateInstanceBinder, Object(), Object)	Provides the implementation for operations that initialize a new instance of a dynamic object. This method is not intended for use in C# or Visual Basic.
	TryDeleteIndex(DeleteIndexBinder, Object())	Provides the implementation for operations that delete an object by index. This method is not intended for use in C# or Visual Basic.
	TryDeleteMember(DeleteMemberBinder)	Provides the implementation for operations that delete an object member. This method is not intended for use in C# or Visual Basic.

	TryGetIndex(GetIndexBinder, Object(), Object)	Provides the implementation for operations that get a value by index. Classes derived from the DynamicObject class can override this method to specify dynamic behavior for indexing operations.
	TryGetMember(GetMemberBinder, Object)	Provides the implementation for operations that get member values. Classes derived from the DynamicObject class can override this method to specify dynamic behavior for operations such as getting a value for a property.
	TryInvoke(InvokeBinder, Object(), Object)	Provides the implementation for operations that invoke an object. Classes derived from the DynamicObject class can override this method to specify dynamic behavior for operations such as invoking an object or a delegate.
	TryInvokeMember(InvokeMemberBinder, Object(), Object)	Provides the implementation for operations that invoke a member. Classes derived from the DynamicObject class can override this method to specify dynamic behavior for operations such as calling a method.
	TrySetIndex(SetIndexBinder, Object(), Object)	Provides the implementation for operations that set a value by index. Classes derived from the DynamicObject class can override this method to specify dynamic behavior for operations that access objects by a specified index.
	TrySetMember(SetMemberBinder, Object)	Provides the implementation for operations that set member values. Classes derived from the DynamicObject class can override this method to specify dynamic behavior for operations such as setting a value for a property.
	TryUnaryOperation(UnaryOperationBinder, Object)	Provides implementation for unary operations. Classes derived from the DynamicObject class can override this method to specify dynamic behavior for operations such as negation, increment, or decrement.

Remarks

The **DynamicObject** class enables you to define which operations can be performed on dynamic objects and how to perform those operations. For example, you can define what happens when you try to get or set an object property, call a method, or perform standard mathematical operations such as addition and multiplication.

This class can be useful if you want to create a more convenient protocol for a library. For example, if users of your library

have to use syntax like `Scriptobj.SetProperty("Count", 1)`, you can provide the ability to use much simpler syntax, like `scriptobj.Count = 1`.

You cannot directly create an instance of the **DynamicObject** class. To implement the dynamic behavior, you may want to inherit from the **DynamicObject** class and override necessary methods. For example, if you need only operations for setting and getting properties, you can override just the TrySetMember and TryGetMember methods.

In C#, to enable dynamic behavior for instances of classes derived from the **DynamicObject** class, you must use the **dynamic** keyword. For more information, see Using Type dynamic (C# Programming Guide).

In Visual Basic, dynamic operations are supported by late binding. For more information, see Early and Late Binding (Visual Basic).

The following code example demonstrates how to create an instance of a class that is derived from the **DynamicObject** class.

VB

```
Public Class SampleDynamicObject
    Inherits DynamicObject
'...
Dim sampleObject As Object = New SampleDynamicObject()
```

You can also add your own members to classes derived from the **DynamicObject** class. If your class defines properties and also overrides the TrySetMember method, the dynamic language runtime (DLR) first uses the language binder to look for a static definition of a property in the class. If there is no such property, the DLR calls the TrySetMember method.

The **DynamicObject** class implements the DLR interface IDynamicMetaObjectProvider, which enables you to share instances of the **DynamicObject** class between languages that support the DLR interoperability model. For example, you can create an instance of the **DynamicObject** class in C# and then pass it to an IronPython function. For more information, see Dynamic Language Runtime Overview and documentation on the CodePlex Web site.

Note

If you have a simple scenario in which you need an object that can only add and remove members at run time but that does not need to define specific operations and does not have static members, use the ExpandoObject class.

If you have a more advanced scenario in which you need to define how dynamic objects participate in the interoperability protocol, or you need to manage DLR fast dynamic dispatch caching, create your own implementation of the IDynamicMetaObjectProvider interface.

Examples

Assume that you want to provide alternative syntax for accessing values in a dictionary, so that instead of writing `sampleDictionary["Text"] = "Sample text"` (`sampleDictionary("Text") = "Sample text"` in Visual Basic), you can write `sampleDictionary.Text = "Sample text"`. Also, you want this syntax to be case-insensitive, so that `sampleDictionary.Text` is equivalent to `sampleDictionary.text`.

The following code example demonstrates the `DynamicDictionary` class, which is derived from the **DynamicObject** class. The `DynamicDictionary` class contains an object of the `Dictionary<string, object>` type (`Dictionary(Of String, Object)` in Visual Basic) to store the key-value pairs, and overrides the TrySetMember and TryGetMember methods to support the new syntax. It also provides a `Count` property, which shows how many dynamic properties the dictionary contains.

VB

```
' The class derived from DynamicObject.
Public Class DynamicDictionary
    Inherits DynamicObject

    ' The inner dictionary.
    Dim dictionary As New Dictionary(Of String, Object)

    ' This property returns the number of elements
    ' in the inner dictionary.
    ReadOnly Property Count As Integer
        Get
            Return dictionary.Count
        End Get
    End Property


    ' If you try to get a value of a property that is
    ' not defined in the class, this method is called.

    Public Overrides Function TryGetMember(
        ByVal binder As System.Dynamic.GetMemberBinder,
        ByRef result As Object) As Boolean

        ' Converting the property name to lowercase
        ' so that property names become case-insensitive.
        Dim name As String = binder.Name.ToLower()

        ' If the property name is found in a dictionary,
        ' set the result parameter to the property value and return true.
        ' Otherwise, return false.
        Return dictionary.TryGetValue(name, result)
    End Function

    Public Overrides Function TrySetMember(
        ByVal binder As System.Dynamic.SetMemberBinder,
        ByVal value As Object) As Boolean

        ' Converting the property name to lowercase
        ' so that property names become case-insensitive.
        dictionary(binder.Name.ToLower()) = value

        ' You can always add a value to a dictionary,
        ' so this method always returns true.
        Return True
    End Function
```

```
End Class

Sub Main()
    ' Creating a dynamic dictionary.
    Dim person As Object = New DynamicDictionary()

    ' Adding new dynamic properties.
    ' The TrySetMember method is called.
    person.FirstName = "Ellen"
    person.LastName = "Adams"

    ' Getting values of the dynamic properties.
    ' The TryGetMember method is called.
    ' Note that property names are now case-insensitive,
    ' although they are case-sensitive in C#.
    Console.WriteLine(person.firstname & " " & person.lastname)

    ' Getting the value of the Count property.
    ' The TryGetMember is not called,
    ' because the property is defined in the class.
    Console.WriteLine("Number of dynamic properties:" & person.Count)

    ' The following statement throws an exception at run time.
    ' There is no "address" property,
    ' so the TryGetMember method returns false and this causes
    ' a MissingMemberException.
    ' Console.WriteLine(person.address)
End Sub
' This examples has the following output:
' Ellen Adams
' Number of dynamic properties: 2
```

For more examples, see Creating Wrappers with DynamicObject on the C# Frequently Asked Questions blog.

Version Information

Universal Windows Platform
Available since 8
.NET Framework
Available since 4.0
Portable Class Library
Supported in: portable .NET platforms
Silverlight
Available since 4.0
Windows Phone Silverlight
Available since 8.0
Windows Phone
Available since 8.1

Thread Safety

Any public static (**Shared** in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

See Also

System.Dynamic Namespace

Return to top

ExpandoObject Class

.NET Framework (current version)

Represents an object whose members can be dynamically added and removed at run time.

Namespace: System.Dynamic
Assembly: System.Core (in System.Core.dll)

Inheritance Hierarchy

System.Object
  System.Dynamic.ExpandoObject

Syntax

VB

```
Public NotInheritable Class ExpandoObject
        Implements IDynamicMetaObjectProvider, IDictionary(Of String, Object),
        ICollection(Of KeyValuePair(Of String, Object)), IEnumerable(Of KeyValuePair(Of
String, Object)),
        IEnumerable, INotifyPropertyChanged
```

Constructors

	Name	Description
	ExpandoObject()	Initializes a new **ExpandoObject** that does not have members.

Methods

	Name	Description
	Equals(Object)	Determines whether the specified object is equal to the current object.(Inherited from Object.)

	GetHashCode()	Serves as the default hash function. (Inherited from Object.)
	GetType()	Gets the Type of the current instance.(Inherited from Object.)
	ToString()	Returns a string that represents the current object.(Inherited from Object.)

Explicit Interface Implementations

	Name	Description
	INotifyPropertyChanged.PropertyChanged	Occurs when a property value changes.
	ICollection(Of KeyValuePair(Of String, Object)).Add(KeyValuePair(Of String, Object))	Adds the specified value to the ICollection(Of T) that has the specified key.
	ICollection(Of KeyValuePair(Of String, Object)).Clear()	Removes all items from the collection.
	ICollection(Of KeyValuePair(Of String, Object)).Contains(KeyValuePair(Of String, Object))	Determines whether the ICollection(Of T) contains a specific key and value.
	ICollection(Of KeyValuePair(Of String, Object)).CopyTo(KeyValuePair(Of String, Object)(), Int32)	Copies the elements of the ICollection(Of T) to an array of type KeyValuePair(Of TKey, TValue), starting at the specified array index.
	ICollection(Of KeyValuePair(Of String, Object)).Remove(KeyValuePair(Of String, Object))	Removes a key and value from the collection.
	IDictionary(Of String, Object).Add(String, Object)	Adds the specified key and value to the dictionary.
	IDictionary(Of String, Object).ContainsKey(String)	Determines whether the dictionary contains the specified key.
	IDictionary(Of String, Object).Remove(String)	Removes the element that has the specified key from the IDictionary.
	IDictionary(Of String, Object).TryGetValue(String, Object)	Gets the value associated with the specified key.

	IEnumerable(Of KeyValuePair(Of String, Object)).GetEnumerator()	Returns an enumerator that iterates through the collection.
	IEnumerable.GetEnumerator()	Returns an enumerator that iterates through the collection.
	IDynamicMetaObjectProvider.GetMetaObject(Expression)	The provided MetaObject will dispatch to the dynamic virtual methods. The object can be encapsulated inside another MetaObject to provide custom behavior for individual actions.
	ICollection(Of KeyValuePair(Of String, Object)).Count	Gets the number of elements in the ICollection(Of T).
	ICollection(Of KeyValuePair(Of String, Object)).IsReadOnly	Gets a value indicating whether the ICollection(Of T) is read-only.
	IDictionary(Of String, Object).Item(String)	Gets or sets the element that has the specified key.
	IDictionary(Of String, Object).Keys	Gets an ICollection(Of T) that contains the keys of the IDictionary(Of TKey, TValue).
	IDictionary(Of String, Object).Values	Gets an ICollection(Of T) that contains the values in the IDictionary(Of TKey, TValue).

Extension Methods

	Name	Description
	Aggregate(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), KeyValuePair(Of String, Object), KeyValuePair(Of String, Object)))	Overloaded. Applies an accumulator function over a sequence.(Defined by Enumerable.)
	Aggregate(Of KeyValuePair(Of String, Object), TAccumulate)(TAccumulate, Func(Of TAccumulate, KeyValuePair(Of String, Object), TAccumulate))	Overloaded. Applies an accumulator function over a sequence. The specified seed value is used as the initial accumulator value.(Defined by Enumerable.)
	Aggregate(Of KeyValuePair(Of String, Object), TAccumulate, TResult)(TAccumulate,	Overloaded. Applies an accumulator function over a sequence. The specified seed value is used

	Func(Of TAccumulate, KeyValuePair(Of String, Object), TAccumulate), Func(Of TAccumulate, TResult))	as the initial accumulator value, and the specified function is used to select the result value.(Defined by Enumerable.)
	All(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Determines whether all elements of a sequence satisfy a condition.(Defined by Enumerable.)
	Any(Of KeyValuePair(Of String, Object))()	Overloaded. Determines whether a sequence contains any elements.(Defined by Enumerable.)
	Any(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Determines whether any element of a sequence satisfies a condition.(Defined by Enumerable.)
	AsEnumerable(Of KeyValuePair(Of String, Object))()	Returns the input typed as IEnumerable(Of T).(Defined by Enumerable.)
	AsParallel()	Overloaded. Enables parallelization of a query. (Defined by ParallelEnumerable.)
	AsParallel(Of KeyValuePair(Of String, Object))()	Overloaded. Enables parallelization of a query. (Defined by ParallelEnumerable.)
	AsQueryable()	Overloaded. Converts an IEnumerable to an IQueryable.(Defined by Queryable.)
	AsQueryable(Of KeyValuePair(Of String, Object))()	Overloaded. Converts a generic IEnumerable(Of T) to a generic IQueryable(Of T).(Defined by Queryable.)
	Average(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Decimal))	Overloaded. Computes the average of a sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Average(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Double))	Overloaded. Computes the average of a sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Average(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Int32))	Overloaded. Computes the average of a sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Average(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Int64))	Overloaded. Computes the average of a sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)

	Average(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Decimal)))	Overloaded. Computes the average of a sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Average(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Double)))	Overloaded. Computes the average of a sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Average(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Int32)))	Overloaded. Computes the average of a sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Average(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Int64)))	Overloaded. Computes the average of a sequence of nullable Int64 values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Average(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Single)))	Overloaded. Computes the average of a sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Average(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Single))	Overloaded. Computes the average of a sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Cast(Of TResult)()	Casts the elements of an IEnumerable to the specified type.(Defined by Enumerable.)
	Concat(Of KeyValuePair(Of String, Object)) (IEnumerable(Of KeyValuePair(Of String, Object)))	Concatenates two sequences.(Defined by Enumerable.)
	Contains(Of KeyValuePair(Of String, Object)) (KeyValuePair(Of String, Object))	Overloaded. Determines whether a sequence contains a specified element by using the default equality comparer.(Defined by Enumerable.)
	Contains(Of KeyValuePair(Of String, Object)) (KeyValuePair(Of String, Object), IEqualityComparer(Of KeyValuePair(Of String, Object)))	Overloaded. Determines whether a sequence contains a specified element by using a specified IEqualityComparer(Of T).(Defined by Enumerable.)
	Count(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the number of elements in a sequence.(Defined by Enumerable.)

	Count(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Returns a number that represents how many elements in the specified sequence satisfy a condition.(Defined by Enumerable.)
	DefaultIfEmpty(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the elements of the specified sequence or the type parameter's default value in a singleton collection if the sequence is empty. (Defined by Enumerable.)
	DefaultIfEmpty(Of KeyValuePair(Of String, Object))(KeyValuePair(Of String, Object))	Overloaded. Returns the elements of the specified sequence or the specified value in a singleton collection if the sequence is empty.(Defined by Enumerable.)
	Distinct(Of KeyValuePair(Of String, Object))()	Overloaded. Returns distinct elements from a sequence by using the default equality comparer to compare values.(Defined by Enumerable.)
	Distinct(Of KeyValuePair(Of String, Object))(IEqualityComparer(Of KeyValuePair(Of String, Object)))	Overloaded. Returns distinct elements from a sequence by using a specified IEqualityComparer(Of T) to compare values. (Defined by Enumerable.)
	ElementAt(Of KeyValuePair(Of String, Object))(Int32)	Returns the element at a specified index in a sequence.(Defined by Enumerable.)
	ElementAtOrDefault(Of KeyValuePair(Of String, Object))(Int32)	Returns the element at a specified index in a sequence or a default value if the index is out of range.(Defined by Enumerable.)
	Except(Of KeyValuePair(Of String, Object))(IEnumerable(Of KeyValuePair(Of String, Object)))	Overloaded. Produces the set difference of two sequences by using the default equality comparer to compare values.(Defined by Enumerable.)
	Except(Of KeyValuePair(Of String, Object))(IEnumerable(Of KeyValuePair(Of String, Object)), IEqualityComparer(Of KeyValuePair(Of String, Object)))	Overloaded. Produces the set difference of two sequences by using the specified IEqualityComparer(Of T) to compare values. (Defined by Enumerable.)
	First(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the first element of a sequence.(Defined by Enumerable.)
	First(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Returns the first element in a sequence that satisfies a specified condition. (Defined by Enumerable.)
	FirstOrDefault(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the first element of a sequence, or a default value if the sequence contains no elements.(Defined by Enumerable.)

	FirstOrDefault(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Returns the first element of the sequence that satisfies a condition or a default value if no such element is found.(Defined by Enumerable.)
	GroupBy(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey))	Overloaded. Groups the elements of a sequence according to a specified key selector function. (Defined by Enumerable.)
	GroupBy(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey), IEqualityComparer(Of TKey))	Overloaded. Groups the elements of a sequence according to a specified key selector function and compares the keys by using a specified comparer. (Defined by Enumerable.)
	GroupBy(Of KeyValuePair(Of String, Object), TKey, TElement)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of KeyValuePair(Of String, Object), TElement))	Overloaded. Groups the elements of a sequence according to a specified key selector function and projects the elements for each group by using a specified function.(Defined by Enumerable.)
	GroupBy(Of KeyValuePair(Of String, Object), TKey, TElement)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of KeyValuePair(Of String, Object), TElement), IEqualityComparer(Of TKey))	Overloaded. Groups the elements of a sequence according to a key selector function. The keys are compared by using a comparer and each group's elements are projected by using a specified function.(Defined by Enumerable.)
	GroupBy(Of KeyValuePair(Of String, Object), TKey, TResult)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of TKey, IEnumerable(Of KeyValuePair(Of String, Object)), TResult))	Overloaded. Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key.(Defined by Enumerable.)
	GroupBy(Of KeyValuePair(Of String, Object), TKey, TResult)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of TKey, IEnumerable(Of KeyValuePair(Of String, Object)), TResult), IEqualityComparer(Of TKey))	Overloaded. Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The keys are compared by using a specified comparer.(Defined by Enumerable.)
	GroupBy(Of KeyValuePair(Of String, Object), TKey, TElement, TResult)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of KeyValuePair(Of String, Object), TElement), Func(Of TKey, IEnumerable(Of TElement), TResult))	Overloaded. Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. The elements of each group are projected by using a specified function.(Defined by Enumerable.)
	GroupBy(Of KeyValuePair(Of String, Object), TKey, TElement, TResult)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of KeyValuePair(Of String, Object), TElement), Func(Of TKey, IEnumerable(Of TElement), TResult), IEqualityComparer(Of TKey))	Overloaded. Groups the elements of a sequence according to a specified key selector function and creates a result value from each group and its key. Key values are compared by using a specified comparer, and the elements of each group are projected by using a specified function.(Defined

		by Enumerable.)
	GroupJoin(Of KeyValuePair(Of String, Object), TInner, TKey, TResult)(IEnumerable(Of TInner), Func(Of KeyValuePair(Of String, Object), TKey), Func(Of TInner, TKey), Func(Of KeyValuePair(Of String, Object), IEnumerable(Of TInner), TResult))	Overloaded. Correlates the elements of two sequences based on equality of keys and groups the results. The default equality comparer is used to compare keys.(Defined by Enumerable.)
	GroupJoin(Of KeyValuePair(Of String, Object), TInner, TKey, TResult)(IEnumerable(Of TInner), Func(Of KeyValuePair(Of String, Object), TKey), Func(Of TInner, TKey), Func(Of KeyValuePair(Of String, Object), IEnumerable(Of TInner), TResult), IEqualityComparer(Of TKey))	Overloaded. Correlates the elements of two sequences based on key equality and groups the results. A specified IEqualityComparer(Of T) is used to compare keys.(Defined by Enumerable.)
	Intersect(Of KeyValuePair(Of String, Object))(IEnumerable(Of KeyValuePair(Of String, Object)))	Overloaded. Produces the set intersection of two sequences by using the default equality comparer to compare values.(Defined by Enumerable.)
	Intersect(Of KeyValuePair(Of String, Object))(IEnumerable(Of KeyValuePair(Of String, Object)), IEqualityComparer(Of KeyValuePair(Of String, Object)))	Overloaded. Produces the set intersection of two sequences by using the specified IEqualityComparer(Of T) to compare values. (Defined by Enumerable.)
	Join(Of KeyValuePair(Of String, Object), TInner, TKey, TResult)(IEnumerable(Of TInner), Func(Of KeyValuePair(Of String, Object), TKey), Func(Of TInner, TKey), Func(Of KeyValuePair(Of String, Object), TInner, TResult))	Overloaded. Correlates the elements of two sequences based on matching keys. The default equality comparer is used to compare keys.(Defined by Enumerable.)
	Join(Of KeyValuePair(Of String, Object), TInner, TKey, TResult)(IEnumerable(Of TInner), Func(Of KeyValuePair(Of String, Object), TKey), Func(Of TInner, TKey), Func(Of KeyValuePair(Of String, Object), TInner, TResult), IEqualityComparer(Of TKey))	Overloaded. Correlates the elements of two sequences based on matching keys. A specified IEqualityComparer(Of T) is used to compare keys.(Defined by Enumerable.)
	Last(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the last element of a sequence.(Defined by Enumerable.)
	Last(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Returns the last element of a sequence that satisfies a specified condition. (Defined by Enumerable.)
	LastOrDefault(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the last element of a sequence, or a default value if the sequence contains no elements.(Defined by Enumerable.)

	LastOrDefault(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Returns the last element of a sequence that satisfies a condition or a default value if no such element is found.(Defined by Enumerable.)
	LongCount(Of KeyValuePair(Of String, Object))()	Overloaded. Returns an Int64 that represents the total number of elements in a sequence.(Defined by Enumerable.)
	LongCount(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Returns an Int64 that represents how many elements in a sequence satisfy a condition.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the maximum value in a generic sequence.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Decimal))	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum Decimal value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Double))	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum Double value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Int32))	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum Int32 value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Int64))	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum Int64 value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Nullable(Of Decimal)))	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum nullable Decimal value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Nullable(Of Double)))	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum nullable Double value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Nullable(Of Int32)))	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum nullable Int32 value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Nullable(Of Int64)))	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum nullable Int64 value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object),	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum

	Nullable(Of Single)))	nullable Single value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Single))	Overloaded. Invokes a transform function on each element of a sequence and returns the maximum Single value.(Defined by Enumerable.)
	Max(Of KeyValuePair(Of String, Object), TResult)(Func(Of KeyValuePair(Of String, Object), TResult))	Overloaded. Invokes a transform function on each element of a generic sequence and returns the maximum resulting value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the minimum value in a generic sequence.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Decimal))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum Decimal value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Double))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum Double value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Int32))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum Int32 value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Int64))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum Int64 value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Decimal)))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum nullable Decimal value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Double)))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum nullable Double value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Int32)))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum nullable Int32 value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Int64)))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum nullable Int64 value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Single)))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum nullable Single value.(Defined by Enumerable.)

	Min(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Single))	Overloaded. Invokes a transform function on each element of a sequence and returns the minimum Single value.(Defined by Enumerable.)
	Min(Of KeyValuePair(Of String, Object), TResult)(Func(Of KeyValuePair(Of String, Object), TResult))	Overloaded. Invokes a transform function on each element of a generic sequence and returns the minimum resulting value.(Defined by Enumerable.)
	OfType(Of TResult)()	Filters the elements of an IEnumerable based on a specified type.(Defined by Enumerable.)
	OrderBy(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey))	Overloaded. Sorts the elements of a sequence in ascending order according to a key.(Defined by Enumerable.)
	OrderBy(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey), IComparer(Of TKey))	Overloaded. Sorts the elements of a sequence in ascending order by using a specified comparer.(Defined by Enumerable.)
	OrderByDescending(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey))	Overloaded. Sorts the elements of a sequence in descending order according to a key.(Defined by Enumerable.)
	OrderByDescending(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey), IComparer(Of TKey))	Overloaded. Sorts the elements of a sequence in descending order by using a specified comparer.(Defined by Enumerable.)
	Reverse(Of KeyValuePair(Of String, Object))()	Inverts the order of the elements in a sequence.(Defined by Enumerable.)
	Select(Of KeyValuePair(Of String, Object), TResult)(Func(Of KeyValuePair(Of String, Object), TResult))	Overloaded. Projects each element of a sequence into a new form.(Defined by Enumerable.)
	Select(Of KeyValuePair(Of String, Object), TResult)(Func(Of KeyValuePair(Of String, Object), Int32, TResult))	Overloaded. Projects each element of a sequence into a new form by incorporating the element's index.(Defined by Enumerable.)
	SelectMany(Of KeyValuePair(Of String, Object), TResult)(Func(Of KeyValuePair(Of String, Object), IEnumerable(Of TResult)))	Overloaded. Projects each element of a sequence to an IEnumerable(Of T) and flattens the resulting sequences into one sequence.(Defined by Enumerable.)
	SelectMany(Of KeyValuePair(Of String, Object), TResult)(Func(Of KeyValuePair(Of String, Object), Int32, IEnumerable(Of TResult)))	Overloaded. Projects each element of a sequence to an IEnumerable(Of T), and flattens the resulting sequences into one sequence. The index of each source element is used in the projected form of that element.(Defined by Enumerable.)

	SelectMany(Of KeyValuePair(Of String, Object), TCollection, TResult)(Func(Of KeyValuePair(Of String, Object), IEnumerable(Of TCollection)), Func(Of KeyValuePair(Of String, Object), TCollection, TResult))	Overloaded. Projects each element of a sequence to an IEnumerable(Of T), flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. (Defined by Enumerable.)
	SelectMany(Of KeyValuePair(Of String, Object), TCollection, TResult)(Func(Of KeyValuePair(Of String, Object), Int32, IEnumerable(Of TCollection)), Func(Of KeyValuePair(Of String, Object), TCollection, TResult))	Overloaded. Projects each element of a sequence to an IEnumerable(Of T), flattens the resulting sequences into one sequence, and invokes a result selector function on each element therein. The index of each source element is used in the intermediate projected form of that element. (Defined by Enumerable.)
	SequenceEqual(Of KeyValuePair(Of String, Object))(IEnumerable(Of KeyValuePair(Of String, Object)))	Overloaded. Determines whether two sequences are equal by comparing the elements by using the default equality comparer for their type.(Defined by Enumerable.)
	SequenceEqual(Of KeyValuePair(Of String, Object))(IEnumerable(Of KeyValuePair(Of String, Object)), IEqualityComparer(Of KeyValuePair(Of String, Object)))	Overloaded. Determines whether two sequences are equal by comparing their elements by using a specified IEqualityComparer(Of T).(Defined by Enumerable.)
	Single(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the only element of a sequence, and throws an exception if there is not exactly one element in the sequence.(Defined by Enumerable.)
	Single(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Returns the only element of a sequence that satisfies a specified condition, and throws an exception if more than one such element exists.(Defined by Enumerable.)
	SingleOrDefault(Of KeyValuePair(Of String, Object))()	Overloaded. Returns the only element of a sequence, or a default value if the sequence is empty; this method throws an exception if there is more than one element in the sequence.(Defined by Enumerable.)
	SingleOrDefault(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Returns the only element of a sequence that satisfies a specified condition or a default value if no such element exists; this method throws an exception if more than one element satisfies the condition.(Defined by Enumerable.)
	Skip(Of KeyValuePair(Of String, Object))(Int32)	Bypasses a specified number of elements in a sequence and then returns the remaining

		elements.(Defined by Enumerable.)
	SkipWhile(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements.(Defined by Enumerable.)
	SkipWhile(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Int32, Boolean))	Overloaded. Bypasses elements in a sequence as long as a specified condition is true and then returns the remaining elements. The element's index is used in the logic of the predicate function.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Decimal))	Overloaded. Computes the sum of the sequence of Decimal values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Double))	Overloaded. Computes the sum of the sequence of Double values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Int32))	Overloaded. Computes the sum of the sequence of Int32 values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Int64))	Overloaded. Computes the sum of the sequence of Int64 values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Decimal)))	Overloaded. Computes the sum of the sequence of nullable Decimal values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Double)))	Overloaded. Computes the sum of the sequence of nullable Double values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object), Nullable(Of Int32)))	Overloaded. Computes the sum of the sequence of nullable Int32 values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object)) (Func(Of KeyValuePair(Of String, Object),	Overloaded. Computes the sum of the sequence of nullable Int64 values that are obtained by

	Nullable(Of Int64)))	invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Nullable(Of Single)))	Overloaded. Computes the sum of the sequence of nullable Single values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Sum(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Single))	Overloaded. Computes the sum of the sequence of Single values that are obtained by invoking a transform function on each element of the input sequence.(Defined by Enumerable.)
	Take(Of KeyValuePair(Of String, Object))(Int32)	Returns a specified number of contiguous elements from the start of a sequence.(Defined by Enumerable.)
	TakeWhile(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Returns elements from a sequence as long as a specified condition is true.(Defined by Enumerable.)
	TakeWhile(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Int32, Boolean))	Overloaded. Returns elements from a sequence as long as a specified condition is true. The element's index is used in the logic of the predicate function.(Defined by Enumerable.)
	ToArray(Of KeyValuePair(Of String, Object))()	Creates an array from a IEnumerable(Of T).(Defined by Enumerable.)
	ToDictionary(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey))	Overloaded. Creates a Dictionary(Of TKey, TValue) from an IEnumerable(Of T) according to a specified key selector function.(Defined by Enumerable.)
	ToDictionary(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey), IEqualityComparer(Of TKey))	Overloaded. Creates a Dictionary(Of TKey, TValue) from an IEnumerable(Of T) according to a specified key selector function and key comparer. (Defined by Enumerable.)
	ToDictionary(Of KeyValuePair(Of String, Object), TKey, TElement)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of KeyValuePair(Of String, Object), TElement))	Overloaded. Creates a Dictionary(Of TKey, TValue) from an IEnumerable(Of T) according to specified key selector and element selector functions.(Defined by Enumerable.)
	ToDictionary(Of KeyValuePair(Of String, Object), TKey, TElement)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of KeyValuePair(Of String, Object), TElement), IEqualityComparer(Of TKey))	Overloaded. Creates a Dictionary(Of TKey, TValue) from an IEnumerable(Of T) according to a specified key selector function, a comparer, and an element selector function.(Defined by Enumerable.)

	ToList(Of KeyValuePair(Of String, Object))()	Creates a List(Of T) from an IEnumerable(Of T).(Defined by Enumerable.)
	ToLookup(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey))	Overloaded. Creates a Lookup(Of TKey, TElement) from an IEnumerable(Of T) according to a specified key selector function.(Defined by Enumerable.)
	ToLookup(Of KeyValuePair(Of String, Object), TKey)(Func(Of KeyValuePair(Of String, Object), TKey), IEqualityComparer(Of TKey))	Overloaded. Creates a Lookup(Of TKey, TElement) from an IEnumerable(Of T) according to a specified key selector function and key comparer.(Defined by Enumerable.)
	ToLookup(Of KeyValuePair(Of String, Object), TKey, TElement)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of KeyValuePair(Of String, Object), TElement))	Overloaded. Creates a Lookup(Of TKey, TElement) from an IEnumerable(Of T) according to specified key selector and element selector functions.(Defined by Enumerable.)
	ToLookup(Of KeyValuePair(Of String, Object), TKey, TElement)(Func(Of KeyValuePair(Of String, Object), TKey), Func(Of KeyValuePair(Of String, Object), TElement), IEqualityComparer(Of TKey))	Overloaded. Creates a Lookup(Of TKey, TElement) from an IEnumerable(Of T) according to a specified key selector function, a comparer and an element selector function.(Defined by Enumerable.)
	Union(Of KeyValuePair(Of String, Object))(IEnumerable(Of KeyValuePair(Of String, Object)))	Overloaded. Produces the set union of two sequences by using the default equality comparer.(Defined by Enumerable.)
	Union(Of KeyValuePair(Of String, Object))(IEnumerable(Of KeyValuePair(Of String, Object)), IEqualityComparer(Of KeyValuePair(Of String, Object)))	Overloaded. Produces the set union of two sequences by using a specified IEqualityComparer(Of T).(Defined by Enumerable.)
	Where(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Boolean))	Overloaded. Filters a sequence of values based on a predicate.(Defined by Enumerable.)
	Where(Of KeyValuePair(Of String, Object))(Func(Of KeyValuePair(Of String, Object), Int32, Boolean))	Overloaded. Filters a sequence of values based on a predicate. Each element's index is used in the logic of the predicate function.(Defined by Enumerable.)
	Zip(Of KeyValuePair(Of String, Object), TSecond, TResult)(IEnumerable(Of TSecond), Func(Of KeyValuePair(Of String, Object), TSecond, TResult))	Applies a specified function to the corresponding elements of two sequences, producing a sequence of the results.(Defined by Enumerable.)

Remarks

The **ExpandoObject** class enables you to add and delete members of its instances at run time and also to set and get values of these members. This class supports dynamic binding, which enables you to use standard syntax like `sampleObject.sampleMember` instead of more complex syntax like `sampleObject.GetAttribute("sampleMember")`.

The **ExpandoObject** class implements the standard Dynamic Language Runtime (DLR) interface IDynamicMetaObjectProvider, which enables you to share instances of the **ExpandoObject** class between languages that support the DLR interoperability model. For example, you can create an instance of the **ExpandoObject** class in C# and then pass it to an IronPython function. For more information, see Dynamic Language Runtime Overview documentation on the CodePlex Web site, and Introducing the ExpandoObject on the C# Frequently Asked Questions Web site.

The **ExpandoObject** class is an implementation of the dynamic object concept that enables getting, setting, and invoking members. If you want to define types that have their own dynamic dispatch semantics, use the DynamicObject class. If you want to define how dynamic objects participate in the interoperability protocol and manage DLR fast dynamic dispatch caching, create your own implementation of the IDynamicMetaObjectProvider interface.

Creating an Instance

In C#, to enable late binding for an instance of the **ExpandoObject** class, you must use the **dynamic** keyword. For more information, see Using Type dynamic (C# Programming Guide).

In Visual Basic, dynamic operations are supported by late binding. For more information, see Early and Late Binding (Visual Basic).

The following code example demonstrates how to create an instance of the **ExpandoObject** class.

VB

```
Dim sampleObject As Object = New ExpandoObject()
```

Adding New Members

You can add properties, methods, and events to instances of the **ExpandoObject** class.

The following code example demonstrates how to add a new property to an instance of the **ExpandoObject** class.

VB

```
sampleObject.Test = "Dynamic Property"
Console.WriteLine(sampleObject.test)
Console.WriteLine(sampleObject.test.GetType())
' This code example produces the following output:
' Dynamic Property
' System.String
```

The methods represent lambda expressions that are stored as delegates, which can be invoked when they are needed. The following code example demonstrates how to add a method that increments a value of the dynamic property.

VB

```
sampleObject.Number = 10
sampleObject.Increment = Function() sampleObject.Number + 1
' Before calling the Increment method.
Console.WriteLine(sampleObject.number)

sampleObject.Increment.Invoke()

' After calling the Increment method.
Console.WriteLine(sampleObject.number)
' This code example produces the following output:
' 10
' 11
```

The following code example demonstrates how to add an event to an instance of the **ExpandoObject** class.

VB

```
Module Module1

Sub Main()
    Dim sampleObject As Object = New ExpandoObject()

    ' Create a new event and initialize it with null.
    sampleObject.sampleEvent = Nothing

    ' Add an event handler.
    Dim handler As EventHandler = AddressOf SampleHandler
    sampleObject.sampleEvent =
        [Delegate].Combine(sampleObject.sampleEvent, handler)

    ' Raise an event for testing purposes.
    sampleObject.sampleEvent.Invoke(sampleObject, New EventArgs())

End Sub

' Event handler.
Sub SampleHandler(ByVal sender As Object, ByVal e As EventArgs)
    Console.WriteLine("SampleHandler for {0} event", sender)
End Sub

' This code example produces the following output:
' SampleHandler for System.Dynamic.ExpandoObject event.

End Module
```

Passing As a Parameter

You can pass instances of the **ExpandoObject** class as parameters. Note that these instances are treated as dynamic objects in C# and late-bound objects in Visual Basic. This means that you do not have IntelliSense for object members

and you do not receive compiler errors when you call non-existent members. If you call a member that does not exist, an exception occurs.

The following code example demonstrates how you can create and use a method to print the names and values of properties.

VB

```
Sub Main()
    Dim employee, manager As Object

    employee = New ExpandoObject()
    employee.Name = "John Smith"
    employee.Age = 33

    manager = New ExpandoObject()
    manager.Name = "Allison Brown"
    manager.Age = 42
    manager.TeamSize = 10

    WritePerson(manager)
    WritePerson(employee)
End Sub

Private Sub WritePerson(ByVal person As Object)

    Console.WriteLine("{0} is {1} years old.",
                      person.Name, person.Age)
    ' The following statement causes an exception
    ' if you pass the employee object.
    ' Console.WriteLine("Manages {0} people", person.TeamSize)

End Sub
```

Enumerating and Deleting Members

The **ExpandoObject** class implements the **IDictionary<String, Object>** interface. This enables enumeration of members added to the instance of the **ExpandoObject** class at run time. This can be useful if you do not know at compile time what members an instance might have.

The following code example shows how you can cast an instance of the **ExpandoObject** class to the IDictionary(Of TKey, TValue) interface and enumerate the instance's members.

VB

```
Dim employee As Object = New ExpandoObject()
employee.Name = "John Smith"
employee.Age = 33
For Each member In CType(employee, IDictionary(Of String, Object))
    Console.WriteLine(member.Key & ": " & member.Value)
Next
```

```
' This code example produces the following output:
' Name: John Smith
' Age: 33
```

In languages that do not have syntax for deleting members (such as C# and Visual Basic), you can delete a member by implicitly casting an instance of the **ExpandoObject** to the **IDictionary<String, Object>** interface and then deleting the member as a key/value pair. This is shown in the following example.

VB

```
Dim employee As Object = New ExpandoObject()
employee.Name = "John Smith"
CType(employee, IDictionary(Of String, Object)).Remove("Name")
```

Receiving Notifications of Property Changes

The **ExpandoObject** class implements the INotifyPropertyChanged interface and can raise a PropertyChanged event when a member is added, deleted, or modified. This enables **ExpandoObject** class integration with Windows Presentation Foundation (WPF) data binding and other environments that require notification about changes in the object content.

The following code example demonstrates how to create an event handler for the `PropertyChanged` event.

VB

```
' Add "Imports System.ComponentModel" line
' to the beginning of the file.
Sub Main()
    Dim employee As Object = New ExpandoObject
    AddHandler CType(
        employee, INotifyPropertyChanged).PropertyChanged,
        AddressOf HandlePropertyChanges
    employee.Name = "John Smith"
End Sub

Private Sub HandlePropertyChanges(
       ByVal sender As Object, ByVal e As PropertyChangedEventArgs)
    Console.WriteLine("{0} has changed.", e.PropertyName)
End Sub
```

Version Information

Universal Windows Platform
Available since 8
.NET Framework
Available since 4.0

Portable Class Library
Supported in: portable .NET platforms
Silverlight
Available since 4.0
Windows Phone Silverlight
Available since 8.0
Windows Phone
Available since 8.1

Thread Safety

Any public static (**Shared** in Visual Basic) members of this type are thread safe. Any instance members are not guaranteed to be thread safe.

See Also

System.Dynamic Namespace

Return to top