Visual Studio 2010 - Visual C#

**dynamic (C# Reference)**

The **dynamic** type enables the operations in which it occurs to bypass compile-time type checking. Instead, these operations are resolved at run time. The **dynamic** type simplifies access to COM APIs such as the Office Automation APIs, and also to dynamic APIs such as IronPython libraries, and to the HTML Document Object Model (DOM).

Type **dynamic** behaves like type **object** in most circumstances. However, operations that contain expressions of type **dynamic** are not resolved or type checked by the compiler. The compiler packages together information about the operation, and that information is later used to evaluate the operation at run time. As part of the process, variables of type **dynamic** are compiled into variables of type **object**. Therefore, type **dynamic** exists only at compile time, not at run time.

The following example contrasts a variable of type **dynamic** to a variable of type **object**. To verify the type of each variable at compile time, place the mouse pointer over dyn or obj in the **WriteLine** statements. IntelliSense shows **dynamic** for dyn and **object** for obj.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl01_code');" \o "Copy Code)

class Program

{

static void Main(string[] args)

{

dynamic dyn = 1;

object obj = 1;

// Rest the mouse pointer over dyn and obj to see their

// types at compile time.

System.Console.WriteLine(dyn.GetType());

System.Console.WriteLine(obj.GetType());

}

}

The **WriteLine** statements display the run-time types of dyn and obj. At that point, both have the same type, integer. The following output is produced:

System.Int32

System.Int32

To see the difference between dyn and obj at compile time, add the following two lines between the declarations and the **WriteLine** statements in the previous example.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl02_code');" \o "Copy Code)

dyn = dyn + 3;

obj = obj + 3;

A compiler error is reported for the attempted addition of an integer and an object in expression obj + 3. However, no error is reported for dyn + 3. The expression that contains dyn is not checked at compile time because the type of dyn is **dynamic**.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifContext

The **dynamic** keyword can appear directly or as a component of a constructed type in the following situations:

* In declarations, as the type of a property, field, indexer, parameter, return value, local variable, or type constraint. The following class definition uses **dynamic** in several different declarations.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_ctl00_ctl00_code');" \o "Copy Code)

class ExampleClass

{

// A dynamic field.

static dynamic field;

// A dynamic property.

dynamic prop { get; set; }

// A dynamic return type and a dynamic paramater type.

public dynamic exampleMethod(dynamic d)

{

// A dynamic local variable.

dynamic local = "Local variable";

int two = 2;

if (d is int)

{

return local;

}

else

{

return two;

}

}

}

* In explicit type conversions, as the target type of a conversion.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_ctl00_ctl01_code');" \o "Copy Code)

static void convertToDynamic()

{

dynamic d;

int i = 20;

d = (dynamic)i;

Console.WriteLine(d);

string s = "Example string.";

d = (dynamic)s;

Console.WriteLine(d);

DateTime dt = DateTime.Today;

d = (dynamic)dt;

Console.WriteLine(d);

}

// Results:

// 20

// Example string.

// 2/17/2009 9:12:00 AM

* In any context where types serve as values, such as on the right side of an **is** operator or an **as** operator, or as the argument to **typeof** as part of a constructed type. For example, **dynamic** can be used in the following expressions.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl24_ctl00_ctl02_code');" \o "Copy Code)

int i = 8;

dynamic d;

// With the is operator.

// The dynamic type behaves like object. The following

// expression returns true unless someVar has the value null.

if (someVar is dynamic) { }

// With the as operator.

d = i as dynamic;

// With typeof, as part of a constructed type.

Console.WriteLine(typeof(List<dynamic>));

// The following statement causes a compiler error.

//Console.WriteLine(typeof(dynamic));

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample

The following example uses **dynamic** in several declarations. The Main method also contrasts compile-time type checking with run-time type checking.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl25_ctl00_ctl00_code');" \o "Copy Code)

using System;

namespace DynamicExamples

{

class Program

{

static void Main(string[] args)

{

ExampleClass ec = new ExampleClass();

Console.WriteLine(ec.exampleMethod(10));

Console.WriteLine(ec.exampleMethod("value"));

// The following line causes a compiler error because exampleMethod

// takes only one argument.

//Console.WriteLine(ec.exampleMethod(10, 4));

dynamic dynamic\_ec = new ExampleClass();

Console.WriteLine(dynamic\_ec.exampleMethod(10));

// Because dynamic\_ec is dynamic, the following call to exampleMethod

// with two arguments does not produce an error at compile time.

// However, itdoes cause a run-time error.

//Console.WriteLine(dynamic\_ec.exampleMethod(10, 4));

}

}

class ExampleClass

{

static dynamic field;

dynamic prop { get; set; }

public dynamic exampleMethod(dynamic d)

{

dynamic local = "Local variable";

int two = 2;

if (d is int)

{

return local;

}

else

{

return two;

}

}

}

}

// Results:

// Local variable

// 2

// Local variable

For more information and examples, see [Using Type dynamic (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/dd264736.aspx).