Visual Studio 2010 - Visual C#

**Boxing and Unboxing (C# Programming Guide)**

Updated: July 2010

Boxing is the process of converting a [value type](http://msdn.microsoft.com/en-us/library/s1ax56ch.aspx) to the type **object** or to any interface type implemented by this value type. When the CLR boxes a value type, it wraps the value inside a System.Object and stores it on the managed heap. Unboxing extracts the value type from the object. Boxing is implicit; unboxing is explicit. The concept of boxing and unboxing underlies the C# unified view of the type system, in which a value of any type can be treated as an object.

In the following example, the integer variable i is *boxed* and assigned to object o.

C#

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

int i = 123;

// The following line boxes i.

object o = i;

The object o can then be unboxed and assigned to integer variable i:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl03_code');" \o "Copy Code)

o = 123;

i = (int)o; // unboxing

The following examples illustrate how boxing is used in C#.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl04_code');" \o "Copy Code)

// String.Concat example.

// String.Concat has many versions. Rest the mouse pointer on

// Concat in the following statement to verify that the version

// that is used here takes three object arguments. Both 42 and

// true must be boxed.

Console.WriteLine(String.Concat("Answer", 42, true));

// List example.

// Create a list of objects to hold a heterogeneous collection

// of elements.

List<object> mixedList = new List<object>();

// Add a string element to the list.

mixedList.Add("First Group:");

// Add some integers to the list.

for (int j = 1; j < 5; j++)

{

// Rest the mouse pointer over j to verify that you are adding

// an int to a list of objects. Each element j is boxed when

// you add j to mixedList.

mixedList.Add(j);

}

// Add another string and more integers.

mixedList.Add("Second Group:");

for (int j = 5; j < 10; j++)

{

mixedList.Add(j);

}

// Display the elements in the list. Declare the loop variable by

// using var, so that the compiler assigns its type.

foreach (var item in mixedList)

{

// Rest the mouse pointer over item to verify that the elements

// of mixedList are objects.

Console.WriteLine(item);

}

// The following loop sums the squares of the first group of boxed

// integers in mixedList. The list elements are objects, and cannot

// be multiplied or added to the sum until they are unboxed. The

// unboxing must be done explicitly.

var sum = 0;

for (var j = 1; j < 5; j++)

{

// The following statement causes a compiler error: Operator

// '\*' cannot be applied to operands of type 'object' and

// 'object'.

//sum += mixedList[j] \* mixedList[j]);

// After the list elements are unboxed, the computation does

// not cause a compiler error.

sum += (int)mixedList[j] \* (int)mixedList[j];

}

// The sum displayed is 30, the sum of 1 + 4 + 9 + 16.

Console.WriteLine("Sum: " + sum);

// Output:

// Answer42True

// First Group:

// 1

// 2

// 3

// 4

// Second Group:

// 5

// 6

// 7

// 8

// 9

// Sum: 30

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

In relation to simple assignments, boxing and unboxing are computationally expensive processes. When a value type is boxed, a new object must be allocated and constructed. To a lesser degree, the cast required for unboxing is also expensive computationally. For more information, see [Performance](http://msdn.microsoft.com/en-us/library/ms173196.aspx).

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

Boxing is used to store value types in the garbage-collected heap. Boxing is an implicit conversion of a [value type](http://msdn.microsoft.com/en-us/library/s1ax56ch.aspx) to the type **object** or to any interface type implemented by this value type. Boxing a value type allocates an object instance on the heap and copies the value into the new object.

Consider the following declaration of a value-type variable:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl40_ctl00_ctl01_code');" \o "Copy Code)

int i = 123;

The following statement implicitly applies the boxing operation on the variable i:

C#

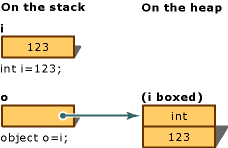
[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl40_ctl00_ctl02_code');" \o "Copy Code)

// Boxing copies the value of i into object o.

object o = i;

The result of this statement is creating an object reference o, on the stack, that references a value of the type **int**, on the heap. This value is a copy of the value-type value assigned to the variable i. The difference between the two variables, i and o, is illustrated in the following figure.

**Boxing Conversion**



It also possible to perform the boxing explicitly as in the following example, but explicit boxing is never required:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl40_ctl00_ctl04_code');" \o "Copy Code)

int i = 123;

object o = (object)i; // explicit boxing

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

This example converts an integer variable i to an object o by using boxing. Then, the value stored in the variable i is changed from 123 to 456. The example shows that the original value type and the boxed object use separate memory locations, and therefore can store different values.

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

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl42_ctl00_ctl00_code');" \o "Copy Code)

class TestBoxing

{

static void Main()

{

int i = 123;

// Boxing copies the value of i into object o.

object o = i;

// Change the value of i.

i = 456;

// The change in i does not effect the value stored in o.

System.Console.WriteLine("The value-type value = {0}", i);

System.Console.WriteLine("The object-type value = {0}", o);

}

}

/\* Output:

The value-type value = 456

The object-type value = 123

\*/

The following example demonstrates a case of invalid unboxing and the resulting **InvalidCastException**. Using **try** and **catch**, an error message is displayed when the error occurs.

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl42_ctl00_ctl01_code');" \o "Copy Code)

class TestUnboxing

{

static void Main()

{

int i = 123;

object o = i; // implicit boxing

try

{

int j = (short)o; // attempt to unbox

System.Console.WriteLine("Unboxing OK.");

}

catch (System.InvalidCastException e)

{

System.Console.WriteLine("{0} Error: Incorrect unboxing.", e.Message);

}

}

}

This program outputs:

Specified cast is not valid. Error: Incorrect unboxing.

If you change the statement:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl42_ctl00_ctl02_code');" \o "Copy Code)

int j = (short) o;

to:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl42_ctl00_ctl03_code');" \o "Copy Code)

int j = (int) o;

the conversion will be performed, and you will get the output:

Unboxing OK.

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

Unboxing is an explicit conversion from the type **object** to a [value type](http://msdn.microsoft.com/en-us/library/s1ax56ch.aspx) or from an interface type to a value type that implements the interface. An unboxing operation consists of:

* Checking the object instance to make sure that it is a boxed value of the given value type.
* Copying the value from the instance into the value-type variable.

The following statements demonstrate both boxing and unboxing operations:

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl43_ctl00_ctl01_code');" \o "Copy Code)

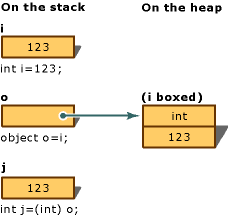
int i = 123; // a value type

object o = i; // boxing

int j = (int)o; // unboxing

The following figure demonstrates the result of the previous statements.

**Unboxing Conversion**



For the unboxing of value types to succeed at run time, the item being unboxed must be a reference to an object that was previously created by boxing an instance of that value type. Attempting to unbox **null** causes a [NullReferenceException](http://msdn.microsoft.com/en-us/library/system.nullreferenceexception.aspx). Attempting to unbox a reference to an incompatible value type causes an [InvalidCastException](http://msdn.microsoft.com/en-us/library/system.invalidcastexception.aspx).