

.NET/C# Syntax.

Modern .NET/C# Course

.NET Academy

Content

- Implicit and Explicit Casting.
- Boxing & Unboxing.

Implicit and Explicit Casting.

Because C# is statically-typed at compile time, after a variable is declared, it cannot be declared again or assigned a value of another type unless that type is implicitly convertible to the variable's type. For example, the string cannot be implicitly converted to int.

Implicit and Explicit Casting.

However, you might sometimes need to copy a value into a variable or method parameter of another type. For example, you might have an integer variable that you need to pass to a method whose parameter is typed as double. Or you might need to assign a class variable to a variable of an interface type. These kinds of operations are called type conversions. In C#, you can perform the following kinds of conversions:

- Implicit conversions: No special syntax is required because the conversion always succeeds and no data will be lost. Examples include conversions from smaller to larger integral types, and conversions from derived classes to base classes.
- Explicit conversions (casts): Explicit conversions require a cast expression. Casting is required when information might be lost in the conversion, or when the conversion might not succeed for other reasons. Typical examples include numeric conversion to a type that has less precision or a smaller range, and conversion of a base-class instance to a derived class.


Implicit and Explicit Casting.

For built-in numeric types, an implicit conversion can be made when the value to be stored can fit into the variable without being truncated or rounded off. For integral types, this means the range of the source type is a proper subset of the range for the target type. For example, a variable of type long (64-bit integer) can store any value that an int (32-bit integer) can store.

```
// Implicit conversion. A long can  
// hold any value an int can hold, and more!  
int num = 2147483647;  
long bigNum = num;
```

Implicit and Explicit Casting.

However, if a conversion cannot be made without a risk of losing information, the compiler requires that you perform an explicit conversion, which is called a cast. A cast is a way of explicitly informing the compiler that you intend to make the conversion and that you are aware that data loss might occur, or the cast may fail at run time. To perform a cast, specify the type that you are casting to in parentheses in front of the value or variable to be converted. The following program casts a double to an int. The program will not compile without the cast.



```
class Test
{
    static void Main()
    {
        double x = 1234.7;
        int a;
        // Cast double to int.
        a = (int)x;
        System.Console.WriteLine(a);
    }
}
// Output: 1234
```

Boxing and Unboxing.

Boxing is the process of converting a value type to the type object or to any interface type implemented by this value type. When the common language runtime (CLR) boxes a value type, it wraps the value inside a System.Object instance 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.



Boxing and Unboxing.

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

```
int i = 123;  
// The following line boxes i.  
object o = i;
```

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

```
o = 123;  
i = (int)o; // unboxing
```


Boxing and Unboxing.

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.

Boxing is used to store value types in the garbage-collected heap. Boxing is an implicit conversion of a value type 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.

Boxing and Unboxing.

Unboxing is an explicit conversion from the type object to a value type 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.



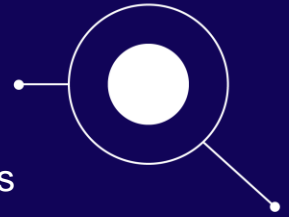
```
int i = 123;      // a value type
object o = i;    // boxing
int j = (int)o;  // unboxing
```

Questions for the study of the topic



1. Which condition in the context of built-in numeric types allows implicit conversion of one type to another?
2. How to perform an explicit conversion in C#? Give an example of converting a value to another type.
3. Why are boxing and unboxing operations computationally intensive, and what makes them less efficient than simple assignments?
4. Describe the steps performed in the unboxing operation in C#, including the checks performed.

Homework



1. What is the main difference between implicit and explicit type conversions in C#?
2. Can you provide an example of an implicit conversion and explain why it doesn't require special syntax?
3. When might you need to perform an explicit conversion (cast) in C#? Provide an example.
4. What is the process of boxing in C#? How is a value type converted into an object or an interface type?
5. Explain the concept of unboxing and how it differs from boxing. Why is unboxing explicit?
6. How does boxing and unboxing relate to the C# unified view of the type system?

Thank you for your attention.

.NET Academy