

Extension Methods

Extension methods allow you to add your own custom method to an existing type. While this seems like a cool feature, there are a few reasons to use extension methods sparingly.

- Adding too many extension methods to an existing type clutters the API.
- If you name your extension method the same as a built-in method yours will never be called.
- If you name your extension method the same as another extension method your method shadows the other extension method.

Because of the above points, you might consider just inheriting from the existing type and adding your own additional methods to this new class. But, with these disclaimers in place, let's learn how you create extension methods because there are cases where using them is perfectly acceptable.

To create extension methods you define a static class with any name you like. Listing 1 shows a class named **StringExtensions**. This listing shows a couple of the available methods such as **ReverseString** and **ToBoolean**. All extension methods must also use the **static** keyword. The first parameter passed to an extension method is the same as the type you are extending and must be prefixed with the keyword **"this"**. Creating classes and methods using these rules informs the compiler to add these methods to the type specified in the first parameter.

```
public static class StringExtensions
{
    public static string Reverse(this string value)
    {
        char[] charArray = null;
        string ret = string.Empty;

        if (value != null)
        {
            charArray = value.ToCharArray();
            Array.Reverse(charArray);

            ret = new string(charArray);
        }

        return ret;
    }

    public static bool ToBoolean(this string value)
    {
        bool ret = false;

        switch (value.ToLower())
        {
            case "true":
            case "t":
            case "yes":
            case "y":
            case "1":
            case "-1":
                ret = true;
                break;
            case "false":
            case "f":
            case "no":
            case "n":
            case "0":
                ret = false;
                break;
            default:
                ret = false;
                break;
        }

        return ret;
    }
}
```

Listing 1: The StringExtensions class extends the string class with additional methods.

To use the methods shown in Listing 1, create an instance of type you are extending. After your new variable type a dot (.) and your extension methods show up in the IntelliSense as shown in Figure 1.

```
private void DemoExtensionMethods()
{
    string value = "Reverse Me";

    value = value.|
}

```

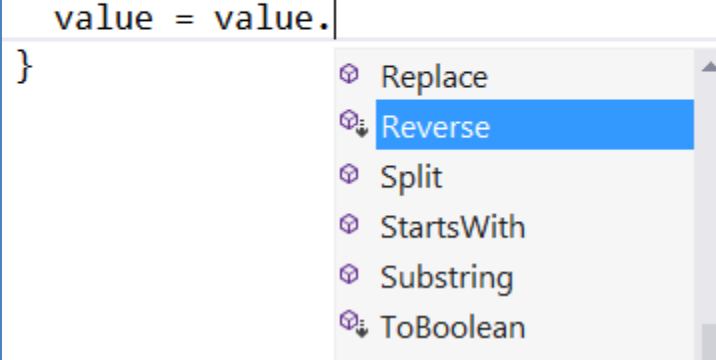
A screenshot of an IDE showing a code completion menu. The code in the background is: `private void DemoExtensionMethods() { string value = "Reverse Me"; value = value.| }`. The cursor is at the end of `value = value.|`. A dropdown menu is open, listing several methods: `Replace`, `Reverse` (highlighted in blue), `Split`, `StartsWith`, `Substring`, and `ToBoolean`. Each method has a small icon to its left.

Figure 1: Using extension methods is the same as any other method on a type.

In the samples that you can download with this article you will find another class that works with the `DateTime` type.