Using C#’s Extension Methods with VBVoice™

Extension Methods are evolving as a powerful feature of the C# programming language for developers who want to extend the functionality of objects for which they do not have the source code. This approach provides methods that resemble original members of the target class, separating it from other approaches to extending functionality. Extension Methods work nicely with Microsoft® Visual Studio®’s Intellisense, a feature of Microsoft’s Visual Studio that offers suggestions on how to complete lines of code, thereby making writing code quicker and more intuitive.

Like any C# class, VBVoice™’s callflow controls and other objects, e.g. *Phrase* and *Greeting*, can be extended with code written by the application developer to perform an action. Using practical examples, this article explains how IVR applications can benefit from Extension Methods. For more details about Extension Methods, refer to C# documentation and the Microsoft website.

Adding *AppendSysPhrase* method to Greeting object:

Building greetings in code is an essential feature of VBVoice. The *Greeting* object provides a method of inserting a system phrase in a particular position within the list of phrases. The system phrase is called from an *EnterEvent* as follows:

```
var gr = e.greeting as Greeting;
var sysPhrase = vbvSysPhraseConstants.vbvSayNumber;
var data1 = “123”;
var data2 = “”;
gr.InsertSysPhrase(index, sysPhrase, data1, data2);
```

When adding multiple consecutive phrases into the same greeting the developer must call this method several times, incrementing the value of index each time or using *gr.Count* as an index when inserting at the end. An extension method called *AppendSysPhrase* eliminates these steps:

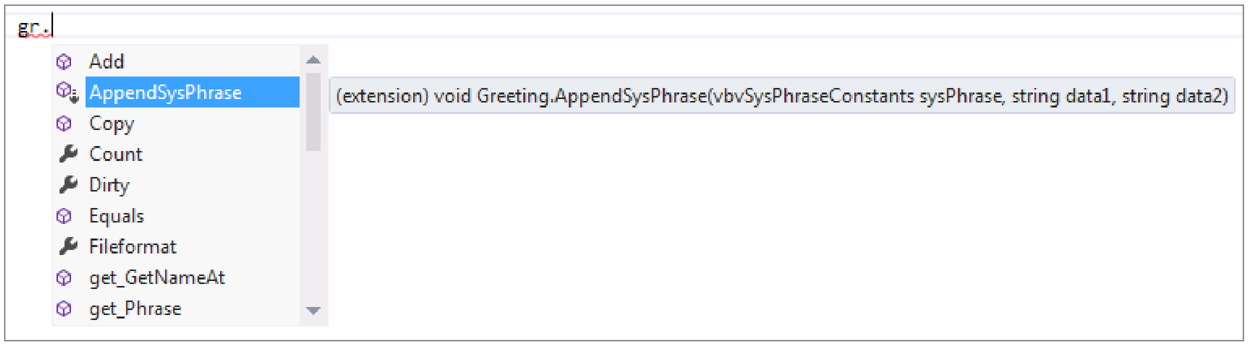
```
namespace Pronexus.VBVoice
{
   public static class MyGreetingEditor
   {
      public static void AppendSysPhrase(this Greeting greeting,
         vbvSysPhraseConstants sysPhrase, string data1, string data2)
      {
         greeting.InsertSysPhrase(greeting.Count, sysPhrase, data1, data2);
      }
   }
}
```

After this code is added to a VBVoice application, the above *EnterEvent* code could be rewritten as follows:

```
gr.AppendSysPhrase(sysPhrase, data1, data1);
```

Notes:

- This is a simplified example but it explains the syntax used to create an extension method.
- The example uses .Net Framework 4.5, but the same principle applies to older frameworks.
- In order for a VBVoice project to use this code, a reference must be added to *VBVoiceSupportLib*.
- The following snapshot shows how Visual Studio picks up the newly added extension in the Intellisense:



```
gr.
```

```
(extension) void Greeting.AppendSysPhrase(vbvSysPhraseConstants sysPhrase, string data1, string data2)
```

Adding *IdleChannelsCount* method to *LineGroup* control:

VBVoice application developers often write routines to search for channels fitting a specific criteria or to count channels that meet certain conditions. The first would be to execute a certain action on that channel; the second is for monitoring or statistical purposes. In this example we will add a method to count the number of channels managed by a certain *LineGroup* that are currently in IDLE state.

To call this method in code, observe the following:

```
var idleChannels = LineGroup1.IdleChannelsCount();
```

Similar to the first example, implementing this method can be done in the following way:

```
namespace Pronexus.VBVoice
{
    public static class MyLineGroupExtensions
    {
        public static int IdleChannelsCount(this LineGroup lineGroup)
        {
            int idlesCount = 0;
            for (short ch = 1; ch <= lineGroup.Channels; ++ch)
            {
                if (lineGroup.LineStatus[ch]
                    == (short)vbvLineStatusConstants.vbvStatusIdle)
                    ++ idlesCount;
            }
            return idlesCount;
        }
    }
}
```

Adding description to extension methods:

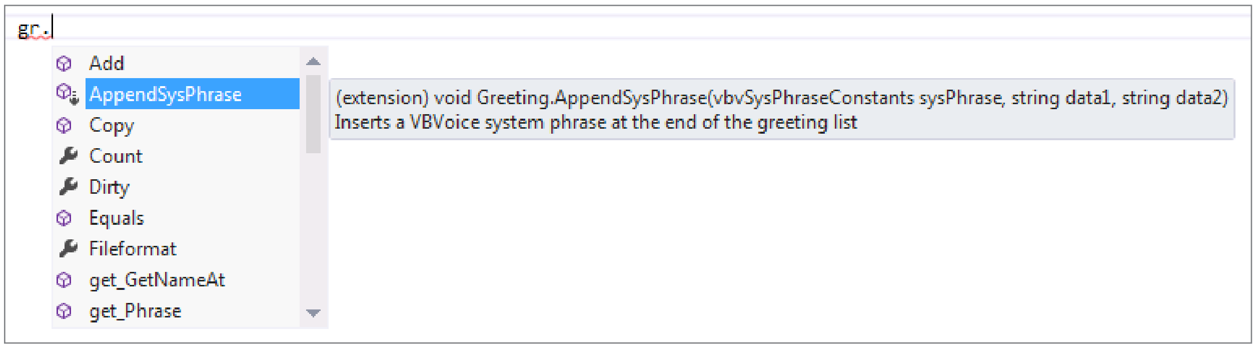
When using extension methods with your application for repetitive and common functions, it is recommended to define them in a separate file; an even better strategy is to keep them in their own library project. This way they can be used with newer projects either by including that file or referencing the dynamic-link library (DLL) built by the library project.

Making these extensions reusable introduces a need for the code documentation and Intellisense description that accompanies other classes and methods in Visual Studio. Just like any other class members in C#, Extension Methods support the use of summary operators to document the usage of the method and its parameters.

To add documentation to the *AppendSysPhrase* method from the first example, add the following code just before the declaration of the method. The example will look as follows:

```
namespace Pronexus.VBVoice
{
   public static class VbvGreetingEditor
   {
      /// <summary>
      /// Inserts a VBVoice system phrase at the end of the greeting list
      /// </summary>
      /// <param name="greeting">VBVoice Greeting object</param>
      /// <param name="sysPhrase">The type of VBVoice system phrase to insert</param>
      /// <param name="data1">Data1 to help create the new phrase</param>
      /// <param name="data2">Data2 to help create the new phrase</param>
      public static void AppendSysPhrase(this Greeting greeting,
         vbvSysPhraseConstants sysPhrase, string data1, string data2)
      {
         greeting.InsertSysPhrase(greeting.Count, sysPhrase, data1, data2);
      }
   }
}
```

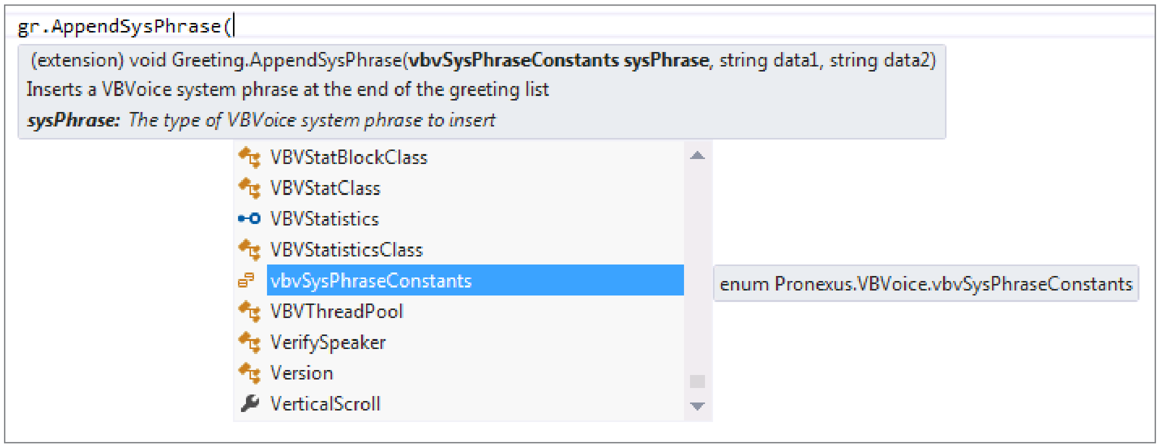
Here is how this will show in Visual Studio's intellisense:



```
gr.
```

(extension) void Greeting.AppendSysPhrase(vbvSysPhraseConstants sysPhrase, string data1, string data2)
Inserts a VBVoice system phrase at the end of the greeting list

and,



```
gr.AppendSysPhrase(
```

(extension) void Greeting.AppendSysPhrase(**vbvSysPhraseConstants sysPhrase**, string data1, string data2)
Inserts a VBVoice system phrase at the end of the greeting list
sysPhrase: *The type of VBVoice system phrase to insert*

enum Pronexus.VBVoice.vbvSysPhraseConstants

Summary:

Although Extension Methods are not the only way of extending functionality to third party components, they serve as excellent vehicles to distribute the new functionality, as they are easier to read and understand than other methods and are therefore more useable. The syntax is straight forward and can easily plug into any project.

About VBVoice

Pronexus VBVoice™ Interactive Voice Response (IVR) toolkit's latest release, VBVoice 8.20, synchronizes all Pronexus' offerings with support for development in Microsoft® Windows® 8, 8.1 and Windows Server® 2012 using the latest Visual Studio® 2013. VBVoice 8.20 includes Microsoft® Lync™ interoperability, secure media and proprietary call control. Download VBVoice IVR toolkit today to discover how easy it is to develop an IVR application.