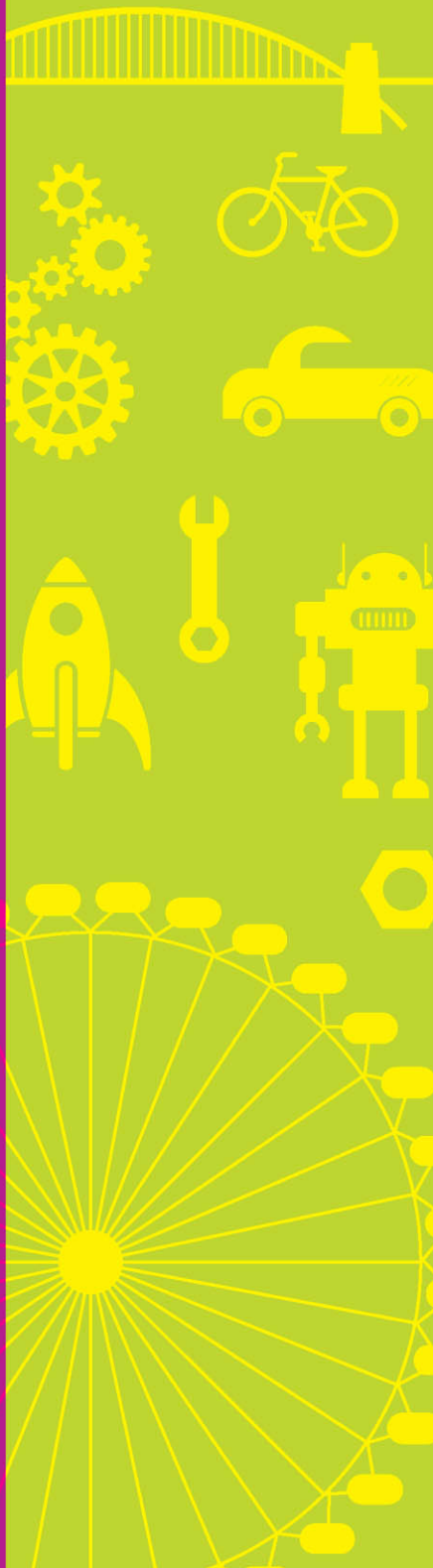
Microsoft

Begin to Code with C#



Rob Miles

Microsoft

Begin to Code with C#

Rob Miles

MAKE SOMETHING HAPPEN

Make some noise

You might go back at this point and add sound effects to the programs that you have already written. The egg-timer program would benefit from an alarm sound, for example. If you can find some suitably eerie background sounds, you could add these to the fortune-teller program.

Displaying image content

You can also display images in your programs. You can display images from the Internet and by using files that are built into the application. The Snaps library provides a method that will fetch and display images from either location. Here is the form of the method that displays an image that is stored with an application:

```
using SnapsLibrary;

class Ch05_17_CityImage
{
    public void StartProgram()
    {
        string url = "ms-appx:///Images/City.jpg";
        SnapsEngine.DisplayImageFromUrl(imageURL: url);
    }
}

Ch05_17_CityImage
```

Create a URL that identifies where the asset can be found.

Display the image

I've called the string variable that holds the name of the image `url` (short for *uniform resource locator*). You may have heard the term URL in the context of webpages. For example, my world famous (in my world) blog has the URL http://www.robmiles.com. The first part of a URL (the part before the // character sequence) is called the *scheme* and describes how to access the data. The scheme "ms-appx" means "look in the content for this asset." The second part of the URL is the actual address of the resource, which in this case is "/Images/City.jpg". If you use Solution Explorer to look in the Images folder, you will find that the folder does contain a file named City.jpg. If you want to add your own images to your applications, you can store them in this folder and then display them by using the `DisplayImageFromUrl` method.

The schemes *http* and *https* mean "look on the Internet and use World Wide Web protocols to find the asset."

```
string url = "https://farm9.staticflickr.com/8713/16988005732_7fefe368cc_d.jpg";
SnapsEngine.DisplayImageFromUrl(url);

Ch05_18_BridgeImage
```

Here you can see how to display an image held in my Flickr account. You can use a statement like this to incorporate pictures from the Internet in your programs (but remember that you must observe any copyright restrictions).

The `DisplayImage` method can use most of the popular image file types, including the JPEG, PNG, GIF and TIFF formats. Keep in mind that if you add a large number of images to your program, it will become larger, because the image files are stored as part of the application. One way to reduce the size of the application is to resize the images. Don't use images directly from your digital camera. Resize these so that they are no more than 1,500 or so pixels wide. This should provide you with enough detail in almost all cases. The best program that I have found for resizing images (and doing lots more besides) is the free image-processing program called Paint.Net, which you can download from http://www.getpaint.net/index.html.

WHAT COULD GO WRONG

Missing files

The `DisplayImageFromUrl` method is prone to failure. It might not be able to load an image because the device the program is running on doesn't have a network connection. Alternatively, the programmer might have mistyped the address of the image in the program. In these situations, the method will not be able to display a picture, which is a problem.

However, the method has been written so that it will not stop the program; instead, it displays a placeholder image to the user that indicates that something went wrong in fetching the image, as you can see here:

You can change this to a message image of your own by replacing the file named ImageNot-Found.png in the Images folder in the BeginToCodeWithCSharp solution.

The `DisplayImage` method returns a `bool` value that indicates whether it was able to display the image that was requested. Here is the code.

```
bool displayedOK = SnapsEngine.DisplayImage(url);
if(displayedOK == false)
{
    SnapsEngine.DisplayString ("Please check your internet connection.");
}
```

If the image is not displayed correctly, the program displays a message prompting the user to check his or her Internet connection. This illustrates an important point about methods when you use them in C# programs: a program does not have to use the result that a method returns. The `DisplayImage` method always returns whether it worked or not, but the first times we used it we ignored this result.

We can simplify the code a bit by using the result from `DisplayImage` directly:

The ! character inverts the Boolean value that follows it.

The DisplayImage method returns true if the image is displayed successfully.

```
if(!SnapsEngine.DisplayImage(url))
{
    SnapsEngine.DisplayString("Please check your internet connection.");
}
```

If you think about it, we want the program to display a message if the image is not displayed—in other words, if `DisplayImage` returns false. We therefore take the result that `DisplayImage` returns and then use the ! operator (not) to invert this.

MAKE SOMETHING HAPPEN

Display some pictures

You can now make programs that display pictures. You can use the `Delay` method to provide a pause between each picture. You could even add some sound effects and use buttons to let the user select the pictures that they want to see. If you display text on the screen, you will find that it is drawn on top of the picture, so you can use this to add captions to the pictures as they are displayed.

What you have learned

In this chapter, you've learned that the C# `if` construction lets you change a program's behavior depending on the data that it is given to work with. This allows a programmer to make software that might be considered "sensible" in that it can respond to input in a useful way.

You also learned that the decision process in C# is based on the Boolean type, which allows programs to work with values that can only be true or false. C# conditions are controlled by the value of Boolean expressions, and the language provides a set of logical operators that can be used in programs to manipulate Boolean values. You can make things happen if two Boolean values are true by using the logical AND operator (`&`). You can also make things happen if one or the other of two values is true by using the logical OR operator (`|`).

You discovered how to write useful programs that work with logical conditions to create code that makes decisions. The best way to implement complex logic is to transcribe a plain description of the decision into C# conditional statements. For example, "If it is Saturday or Sunday and it is after 9:00 a.m., I must get out of bed" could be converted to a single logical expression that makes that decision.

Finally, you have seen how to add and use assets in a program, making use of Visual Studio to manage the asset files and Snaps methods that let you use the images and

sound files. You have also seen how a unified resource locator string allows a program to load assets from the Internet.

Here are some questions that you might like to ponder about the process of making decisions in programs:

Does the use of Boolean values mean that a program will always do the same thing given the same data inputs?

It is very important that, given the same inputs, the computer does exactly the same thing each time. If the computer starts to behave in an inconsistent way, this makes it much less useful. When we want random behavior from a computer (for example, when we are playing a game against a computer opponent), we have to obtain values that are explicitly random and make decisions based on those. Nobody wants a "moody" computer that changes its mind (although, of course, it might be fun to try and program one by using random numbers).

Will the computer always do the right thing when we write programs that make decisions?

It would be great if we could guarantee that the computer will always do the right thing. However, the computer is only ever as good as the program that it is running. If something happens that the program was not expecting, this can cause it to do the wrong thing in response. For example, if a program was working out the cooking time for a bowl of soup and the user entered 10 servings rather than 1, the program would set the cooking time to be far too long (and probably burn down the kitchen in the process). In that situation, you can blame the user (because he put in the wrong data), but there should probably also be a test in the program that checks to see whether the value entered was sensible. If the cooker can't actually hold more than three servings, it would seem sensible to perform a test that limits the input to three. When you write a program, you need to anticipate what the user might do and create decisions that make your program behave sensibly in each situation.

6

Repeating actions with loops