**Constants vs. Read-Only Fields**

There will certainly be times when you have fields that you do not want altered during the execution of the application—for example, data files your application depends on, the value of *pi* for a math class, or any value that you use in your application that you know will never change. To address these situations, C# allows for the definition of two closely related member types: constants and read-only fields.

## Constants

As you can guess from the name, *constants*—represented by the *const* keyword—are fields that remain constant for the life of the application. There are only two rules to keep in mind when defining something as a *const*. First, a constant is a member whose value is set at compile time, either by the programmer or defaulted by the compiler. Second, a constant member's value must be written as a literal.

To define a field as a constant, specify the *const* keyword before the member being defined, as follows:

using System;

class MagicNumbers

{

**public const double pi = 3.1415;**

**public const int answerToAllLifesQuestions = 42;**

}

class ConstApp

{

public static void Main()

{

Console.WriteLine("pi = {0}, everything else = {1}",

MagicNumbers.pi, MagicNumbers.answerToAllLifesQuestions);

}

}

Notice one key point about this code. There's no need for the client to instantiate the *MagicNumbers* class because by default *const* members are static

## Read-Only Fields

A field defined as a *const* is useful because it clearly documents the programmer's intention that the field contain an immutable value. However, that only works if you know the value at compile time. So what does a programmer do when the need arises for a field with a value that won't be known until run time and should not be changed once it's been initialized? This issue—typically not addressed in other languages—was resolved by the designers of the C# language with what's called a *read-only field*.

When you define a field with the *readonly* keyword, you have the ability to set that field's value in one place: the constructor. After that point, the field cannot be changed by either the class itself or the class's clients. Let's say you want to keep track of the screen's resolution for a graphics application. You can't address this problem with a *const* because the application cannot determine the end user's screen resolution until run time, so you use code like that at the top of the following page.

using System;

class GraphicsPackage

{

**public readonly int ScreenWidth;**

**public readonly int ScreenHeight;**

public GraphicsPackage()

{

this.ScreenWidth = 1024;

this.ScreenHeight = 768;

}

}

class ReadOnlyApp

{

public static void Main()

{

GraphicsPackage graphics = new GraphicsPackage();

Console.WriteLine("Width = {0}, Height = {1}",

graphics.ScreenWidth,

graphics.ScreenHeight);

}

}

At first glance, this code seems to be just what you would need. However, there's one small issue: the read-onlyfields that we defined are instancefields, meaning that the user would have to instantiate the class to use the fields. This might not be a problem and could even be what you want in cases in which the way the class is instantiated will determine the read-onlyfield's value. But what if you want a constant, which is static by definition, that can be initialized at run time? In that case, you would define the field with both the *static* and the *readonly* modifiers. Then you'd create a special type of constructor called a *static constructor.* **Static constructors are constructors that are used to initialize static fields, read-only** or otherwise. Here I've modified the previous example to make the screen resolution fields static and read-only, and I've added a static constructor. Note the addition of the *static* keyword to the constructor's definition:

using System;

class GraphicsPackage

{

public static readonly int ScreenWidth;

public static readonly int ScreenHeight;

**static GraphicsPackage()**

{

// Code would be here to

// calculate resolution.

ScreenWidth = 1024;

ScreenHeight = 768;

}

}

class ReadOnlyApp

{

public static void Main()

{

Console.WriteLine("Width = {0}, Height = {1}",

GraphicsPackage.ScreenWidth,

GraphicsPackage.ScreenHeight);

}

}