Working with Constant Field Data

C# offers the const keyword to define constant data, which can never change after the initial assignment.

As you might guess, this can be helpful when you are defining a set of known values for use in your

applications that are logically connected to a given class or structure.

Assume you are building a utility class named MyMathClass that needs to define a value for the value

PI (which you will assume to be 3.14). Begin by creating a new Console Application project named

ConstData. Given that you would not want to allow other developers to change this value in code, PI

could be modeled with the following constant:

namespace ConstData

{

class MyMathClass

{

public const double PI = 3.14;

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("\*\*\*\*\* Fun with Const \*\*\*\*\*\n");

Console.WriteLine("The value of PI is: {0}", MyMathClass.PI);

// Error! Can't change a constant!

// MyMathClass.PI = 3.1444;

Console.ReadLine();

}

}

}

Notice that you are referencing the constant data defined by MyMathClass using a class name prefix

**(i.e., MyMathClass.PI). This is due to the fact that constant fields of a class are implicitly *static***. However,

it is permissible to define and access a local constant variable within a type member. By way of example:

static void LocalConstStringVariable()

{

// A local constant data point can be directly accessed.

const string fixedStr = "Fixed string Data";

Console.WriteLine(fixedStr);

// Error!

fixedStr = "This will not work!";

}

Regardless of where you define a constant piece of data, the one point to always remember is that

**the initial value assigned to the constant must be specified at the time you define the constant**.

Thus, if you were to modify your MyMathClass in such a way that the value of PI is assigned in a class constructor

as follows:

class MyMathClass

{

// Try to set PI in ctor?

public const double PI;

public MyMathClass()

{

// Error!

PI = 3.14;

}

}

you would receive a compile-time error. The reason for this restriction has to do with the fact the value

of constant data must be known at compile time. Constructors, as you know, are invoked at runtime.

Understanding Read-Only Fields

Closely related to constant data is the notion of *read-only field data* (which should not be confused with a read-only property). Like a constant, a **read-only field cannot be changed after the initial assignment**.

However, unlike a constant, the value assigned to a read-only field can be determined at runtime and, therefore, can legally be assigned within **the scope of a constructor, but nowhere else.**

This can be very helpful when you don’t know the value of a field until runtime, perhaps because

you need to read an external file to obtain the value, but wish to ensure that the value will not change

after that point.

using System;

namespace ConstData

{

classMyMathClass

{

Public readonly double PI;

public MyMathClass (double d)

{

PI = d;

}

}

classProgram

{

staticvoid Main(string[] args)

{

Console.WriteLine("\*\*\*\*\* Fun with Const \*\*\*\*\*\n");

// Error! Can't change a constant!

// MyMathClass.PI = 3.1444;

MyMathClass mm = newMyMathClass(3.14);

Console.WriteLine("{0}", mm.PI);

Console.ReadLine();

}

}

}

Static Read-Only Fields

Unlike a constant field, read-only fields are not implicitly static. Thus, if you want to expose PI from the class level, you must explicitly make use of the static keyword.

If you know the value of a static readonly field at compile time, the initial assignment looks very similar to that of a constant (however in this case, it would be easier to simply use the const keyword in the first place, as we are assigning the data field at the time of declaration):

class MyMathClass

{

public static readonly double PI = 3.14;

}

class Program

{

static void Main(string[] args)

{

Console.WriteLine("\*\*\*\*\* Fun with Const \*\*\*\*\*");

Console.WriteLine("The value of PI is: {0}", MyMathClass.PI);

Console.ReadLine();

}

}

However, if the value of a static read-only field is not known until runtime, you must make use of a

**static constructor**as described

class MyMathClass

{

public static readonly double PI;

static MyMathClass()

{ PI = 3.14; }

}