**Introducing enums**

An enumeration type, or enum, is a structure that enables you to create a variable with a fixed set of possible values. The most common example is to use an enum to define the day of the week. There are only seven possible values for days of the week, and you can be reasonably certain that these values will never change.

A best practice would be to define your enum directly within a namespace so that all classes in that namespace will have access to it, if needed. You can also be nest your enums within classes or structs.  
  
By default enum values start at 0 and each successive member is increased by a value of 1.

**Creating and Using Enums**

To create an enum, you declare it in your code file with the following syntax, which demonstrates creating an enum called Day, that contains the days of the week:

enum Day { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

By default enum values start at 0 and each successive member is increased by a value of 1.  As a result, the previous enum 'Day' would contain the values:

* Sunday = 0
* Monday = 1
* Tuesday = 2
* etc.

You can change the default by specifying a starting value for your enum as in the following example.

enum Day { Sunday = 1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

In this example, Sunday is given the value 1 instead of the detaul 0.  Now Monday is 2, Tuesday is 3, etc.

The keyword enum is used to specify the "type" that the variable Day will be.  In this case, an enumeration type.  Enums support intrinsic data types and can be any one of the following:  
  
- byte  
- sbyte  
- short  
- ushort  
- int  
- uint  
- long  
- ulong

In order to change the default data type of your enum, you precede the list with a data type from the list above, such as:

enum Day : short { Sunday = 1, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday };

The underlying type specifies how much storage will be allocated for each enumerator in the enum.  During compile time, your enum will be converted to numeric literals in your code.  If you are using Visual Studio, the Intellisense feature is fully capable of recognizing your enums and will display the string values automatically in the IDE as you type the enum name.

It's important to note that you will be required to use an explicit cast if you want to convert from an enum type to an integral type. Consider this example where the statement assigns the enumerator Sun to an int type, with a cast, to convert from enum to int.  
  
int x = (int)Days.Sun;

**Using an Enum**

Day favoriteDay = Day.Friday;

Using enums has several advantages over using text or numerical types: - Improved manageability. By constraining a variable to a fixed set of valid values, you are less likely to experience invalid arguments and spelling mistakes. - Improved developer experience. In Visual Studio, the IntelliSense feature will prompt you with the available values when you use an enum. - Improved code readability. The enum syntax makes your code easier to read and understand.

Each member of an enum has a name and a value. The name is the string you define in the braces, such as Sunday or Monday. By default, the value is an integer. If you do not specify a value for each member, the members are assigned incremental values starting with 0. For example, Day.Sunday is equal to 0 and Day.Monday is equal to 1.

The following example shows how you can use names and values interchangeably:

**Using Enum Names and Values Interchangeably**

// Set an enum variable by name.  
Day favoriteDay = Day.Friday;  
// Set an enum variable by value.   
Day favoriteDay = (Day)4;