Stefan Djokic

[Flags]

with

Enums

What is that?

Purpose of [Flags]

The [Flags] attribute indicates that an enum should be treated as a set of flags.

```
[Flags]
enum AccessRights
{
    None    = 0,    // 0000
    Read    = 1,    // 0001
    Write   = 2,    // 0010
    Execute = 4     // 0100
}
```

Each value in this enum is assigned a power of 2, so each value corresponds to a different bit being set in the number's binary representation.

Combining Flags

To combine flags, you'd typically use the bitwise OR | operation:

```
AccessRights readWrite = AccessRights.Read | AccessRights.Write;
// Result: 0011 in binary (which is 3 in decimal)
```

Checking Flags

To check if a particular flag is set, you'd use the bitwise AND & operation:

```
bool hasReadAccess = (readWrite & AccessRights.Read) == AccessRights.Read;
// Result: true, because 0011 AND 0001 is 0001
```

Removing Flags

To remove a flag, you can use the bitwise AND operation combined with the bitwise NOT ~ operation:

```
readWrite = readWrite & ~AccessRights.Read;
// Result: 0010 in binary (which is 2, representing Write access)
```

[Flags] Attribute and ToString()

The [Flags] attribute also changes the behavior of the ToString() method on the enum. Without the [Flags] attribute, ToString() just gives the name of the value. With the [Flags] attribute, if the value is a combination of flags, ToString() gives a string that represents the combination:

```
AccessRights myRights = AccessRights.Read | AccessRights.Write;
Console.WriteLine(myRights);  // Outputs "Read, Write"
```

Best Practices

1. Always include a 'None' value set to 0. This represents the absence of any flags.

2. Assign powers of 2 as values to ensure each value corresponds to a unique bit.

3. Only use the [Flags] attribute when you actually intend the enum to represent multiple values simultaneously.

DO YOU LIKE THE POST?

REPOST IT!