## **Enums: Named Numeric ConstantsEnums: Konstanta Numerik Bernama**

An **enum** (short for "enumeration") is a distinct value type that allows you to define a group of named integral constants. Instead of using "magic numbers" (raw numbers with implicit meaning), enums provide more readable and maintainable code by associating names with these numeric values.Enum (singkatan dari “enumeration”) adalah tipe nilai unik yang memungkinkan Anda mendefinisikan kelompok konstanta bilangan bulat yang diberi nama. Alih-alih menggunakan “angka ajaib” (angka mentah dengan makna implisit), enum menyediakan kode yang lebih mudah dibaca dan dirawat dengan mengaitkan nama-nama tersebut dengan nilai-nilai numerik.

For example:

|  |
| --- |
| public enum BorderSide { Left, Right, Top, Bottom } |

You can then use this enum type in your code:

|  |
| --- |
| BorderSide topSide = BorderSide.Top; // Assigning an enum member bool isTop = (topSide == BorderSide.Top); // Comparing enum values (output: true) |

### **Underlying Integral Values**Nilai-nilai Integral yang Mendasar

Each member of an enum has an **underlying integral value**. By default:Setiap anggota enum memiliki nilai integer yang mendasarinya. Secara default:

* The underlying type is int.Tipe dasarnya adalah int.
* The constants are automatically assigned values starting from 0, incrementing by 1 for each subsequent member in the declaration order.Konstanta-konstanta secara otomatis diberi nilai mulai dari 0, bertambah 1 untuk setiap anggota berikutnya dalam urutan deklarasi.

So, for BorderSide, the default values would be: Left = 0, Right = 1, Top = 2, Bottom = 3.Jadi, untuk BorderSide, nilai defaultnya adalah: Kiri = 0, Kanan = 1, Atas = 2, Bawah = 3.

You can **specify an alternative integral type** for the enum, such as byte, short, long, etc.:Anda dapat menentukan tipe integral alternatif untuk enum, seperti byte, short, long, dan sebagainya:

|  |
| --- |
| public enum BorderSide : byte { Left, Right, Top, Bottom } // Underlying type is byte |

You can also **explicitly assign specific underlying values** to enum members:Anda juga dapat secara eksplisit menetapkan nilai dasar tertentu ke anggota enum:

|  |
| --- |
| public enum BorderSide : byte { Left = 1, Right = 2, Top = 10, Bottom = 11 } |

The compiler also allows you to explicitly assign values to only *some* members. Any unassigned members will continue incrementing from the last explicit value.Penerjemah juga memungkinkan Anda untuk secara eksplisit menetapkan nilai hanya pada beberapa anggota. Anggota yang tidak ditetapkan nilainya akan terus bertambah dari nilai eksplisit terakhir.

|  |
| --- |
| public enum BorderSide : byte {  Left = 1,  Right, // Automatically assigned 2  Top = 10,  Bottom // Automatically assigned 11 } |

## **Enum Conversions**

You can convert an enum instance to and from its underlying integral value using an **explicit cast**:Anda dapat mengonversi instance enum menjadi dan dari nilai integral dasarnya menggunakan casting eksplisit:

|  |
| --- |
| int i = (int)BorderSide.Left; // i will be 1 (assuming Left=1) BorderSide side = (BorderSide)i; // side will be BorderSide.Left bool leftOrRight = (int)side <= 2; // Converts 'side' to its int value for comparison |

You can also explicitly cast one enum type to another, provided their underlying integral values are compatible. The conversion essentially happens via the underlying numeric values.Anda juga dapat secara eksplisit mengubah tipe enum satu ke tipe enum lainnya, asalkan nilai integer dasarnya kompatibel. Konversi ini pada dasarnya dilakukan melalui nilai numerik dasarnya.

|  |
| --- |
| public enum HorizontalAlignment {  Left = BorderSide.Left, // Assigns 1  Right = BorderSide.Right, // Assigns 2  Center // Automatically assigned 3 }  HorizontalAlignment h = (HorizontalAlignment)BorderSide.Right; // h will be HorizontalAlignment.Right // This is effectively: HorizontalAlignment h = (HorizontalAlignment)(int)BorderSide.Right; |

### **Special Treatment of the Numeric Literal 0 Perlakuan Khusus terhadap Bilangan Literasi 0**

The numeric literal 0 is treated specially by the compiler in enum expressions and **does not require an explicit cast**:Nilai literal numerik 0 diperlakukan secara khusus oleh compiler dalam ekspresi enum dan tidak memerlukan casting eksplisit:

|  |
| --- |
| BorderSide b = 0; // OK, no cast needed. 'b' will have the underlying value 0. if (b == 0) { /\* ... \*/ } // OK |

This special treatment is due to two common conventions:Perlakuan khusus ini disebabkan oleh dua konvensi umum:

* The first member of an enum (which defaults to 0 if unassigned) is often used as a "default" or "none" value.Anggota pertama dari sebuah enum (yang secara default bernilai 0 jika tidak ditentukan) sering digunakan sebagai nilai “default” atau “tidak ada”.
* For flag enums (discussed next), 0 typically represents "no flags."Untuk enum bendera (yang akan dibahas selanjutnya), 0 biasanya mewakili “tidak ada bendera.”

## **Flags Enums**

**Flags enums** (also known as bit field enums) allow you to combine multiple enum members using bitwise operators. This is particularly useful for representing a set of options or permissions.Enum bendera (juga dikenal sebagai enum bidang bit) memungkinkan Anda menggabungkan beberapa anggota enum menggunakan operator bitwise. Hal ini sangat berguna untuk mewakili sekumpulan opsi atau izin.

To prevent ambiguities, members of a combinable enum **must have explicitly assigned values**, typically in powers of two. This ensures that each combination results in a unique underlying integral value.Untuk menghindari ambiguitas, anggota dari enum yang dapat digabungkan harus memiliki nilai yang secara eksplisit ditetapkan, biasanya dalam kelipatan dua. Hal ini memastikan bahwa setiap kombinasi menghasilkan nilai integer dasar yang unik.

|  |
| --- |
| [Flags] // The [Flags] attribute is crucial for proper ToString() behavior enum BorderSides {  None = 0,  Left = 1, // 0001  Right = 2, // 0010  Top = 4, // 0100  Bottom = 8 // 1000 } // Or using bit shifts for readability: // enum BorderSides { None=0, Left=1, Right=1<<1, Top=1<<2, Bottom=1<<3 } |

You use **bitwise operators** (| for OR, & for AND, ^ for XOR, ~ for NOT) to combine and test these enum values. These operations act on the underlying integral values.Anda menggunakan operator bitwise (| untuk OR, & untuk AND, ^ untuk XOR, ~ untuk NOT) untuk menggabungkan dan menguji nilai-nilai enum ini. Operasi-operasi ini bekerja pada nilai-nilai integer yang mendasarinya.

|  |
| --- |
| BorderSides leftRight = BorderSides.Left | BorderSides.Right; // Combines Left and Right (1 | 2 = 3)  if ((leftRight & BorderSides.Left) != 0) // Check if 'Left' is included  Console.WriteLine("Includes Left"); // Output: Includes Left  string formatted = leftRight.ToString(); // Output: "Left, Right" (thanks to [Flags] attribute)  BorderSides s = BorderSides.Left; s |= BorderSides.Right; // Add Right to s (s now holds Left | Right) Console.WriteLine(s == leftRight); // Output: True  s ^= BorderSides.Right; // Toggle BorderSides.Right (XOR with Right) Console.WriteLine(s); // Output: Left (Right bit is now off) |

**Convention:**

* Always apply the [Flags] attribute to an enum type when its members are combinable. Without it, ToString() on a combined value will just emit the numeric value (e.g., "3") instead of the named members ("Left, Right").Selalu terapkan atribut [Flags] pada tipe enum ketika anggotanya dapat digabungkan. Tanpa atribut ini, metode ToString() pada nilai yang digabungkan hanya akan menampilkan nilai numerik (misalnya, “3”) alih-alih anggota yang diberi nama (“Left, Right”).
* Combinable enum types are conventionally given a **plural name** (e.g., BorderSides).Jenis enum yang dapat digabungkan secara konvensional diberi nama jamak (misalnya, BorderSides).

For convenience, you can include common combinations as members within the enum declaration itself:Untuk kemudahan, Anda dapat menyertakan kombinasi umum sebagai anggota dalam deklarasi enum itu sendiri:

|  |
| --- |
| [Flags] enum BorderSides {  None = 0,  Left = 1, Right = 1 << 1, Top = 1 << 2, Bottom = 1 << 3,  LeftRight = Left | Right,  TopBottom = Top | Bottom,  All = LeftRight | TopBottom } |

## **Enum Operators**

Most standard operators can be used with enums. They generally operate on the enum's underlying integral values:Sebagian besar operator standar dapat digunakan dengan enum. Operator-operator ini umumnya beroperasi pada nilai integer dasar dari enum:

* **Assignment:** =
* **Equality/Inequality:** ==, !=
* **Comparison:** <, >, <=, >=
* **Arithmetic:** +, -, ++, -- (additions/subtractions between an enum and an integral type are allowed, but not between two enums).
* **Bitwise:** ^ (XOR), & (AND), | (OR), ~ (NOT)
* **Compound Assignment:** +=, -=
* **Size:** sizeof (returns the size of the underlying type)

The bitwise, arithmetic, and comparison operators will perform their operations on the underlying integral values of the enum.Operator bitwise, aritmatika, dan perbandingan akan melakukan operasinya pada nilai integer dasar dari enum.

## **Type-Safety Issues**

While enums enhance type safety, it's important to be aware of some potential issues stemming from their close relationship with integral types.Meskipun enums meningkatkan keamanan tipe, penting untuk menyadari beberapa masalah potensial yang timbul dari hubungan eratnya dengan tipe integral.

Because an enum can be explicitly cast to and from its underlying integral type, it's possible to assign a numeric value that does not correspond to any defined enum member:Karena enum dapat secara eksplisit dikonversi ke dan dari tipe integral dasarnya, dimungkinkan untuk mengisi nilai numerik yang tidak sesuai dengan anggota enum yang telah didefinisikan:

|  |
| --- |
| public enum BorderSide { Left, Right, Top, Bottom } BorderSide b = (BorderSide)12345; // No compile-time error Console.WriteLine(b); // Output: 12345 (just prints the numeric value) |

Similarly, bitwise or arithmetic operators can also produce "invalid" enum values.Demikian pula, operator aritmatika bitwise OR juga dapat menghasilkan nilai enum yang “tidak valid”.

|  |
| --- |
| This can lead to issues if your code assumes that an enum variable will always hold a valid, named member. For example: void Draw(BorderSide side) {  if (side == BorderSide.Left) { /\* ... \*/ }  else if (side == BorderSide.Right) { /\* ... \*/ }  else if (side == BorderSide.Top) { /\* ... \*/ }  else { /\* ... \*/ } // Assumes BorderSide.Bottom, but could be 12345! } |

To address this, you should:

* **Add defensive checks:** Include an else clause to handle unexpected values, potentially throwing an ArgumentException.Tambahkan pemeriksaan defensif: Tambahkan klausa else untuk menangani nilai yang tidak terduga, yang berpotensi memicu pengecualian ArgumentException.
* **Validate the enum value:** The static Enum.IsDefined method can check if a given integral value corresponds to a named member in a *non-flags* enum.Memvalidasi nilai enum: Metode statis Enum.IsDefined dapat memeriksa apakah nilai integer yang diberikan sesuai dengan anggota yang diberi nama dalam enum non-flags.

|  |
| --- |
| BorderSide side = (BorderSide)12345; Console.WriteLine(Enum.IsDefined(typeof(BorderSide), side)); // Output: False |

* However, Enum.IsDefined does **not** work correctly for [Flags] enums because a combined flag value (e.g., Left | Right) is not considered a "defined" single member. For flags enums, a common trick is to use Enum.ToString() and decimal.TryParse() as a heuristic to check validity, as shown in the example provided in your document.Namun, metode Enum.IsDefined tidak berfungsi dengan benar untuk enum [Flags] karena nilai bendera gabungan (misalnya, Left | Right) tidak dianggap sebagai anggota tunggal yang “didefinisikan”. Untuk enum bendera, trik umum adalah menggunakan Enum.ToString() dan decimal.TryParse() sebagai heuristik untuk memeriksa validitas, seperti yang ditunjukkan dalam contoh yang disediakan dalam dokumen Anda.

Enums are a valuable tool for creating clear, readable, and type-safe code when dealing with fixed sets of choices. Understanding their underlying integral nature and potential type-safety pitfalls allows for their effective and robust use.Enums adalah alat yang berguna untuk membuat kode yang jelas, mudah dibaca, dan aman tipe saat bekerja dengan kumpulan pilihan yang tetap. Memahami sifat integral dasarnya dan potensi jebakan keamanan tipe memungkinkan penggunaan yang efektif dan andal.