

[< Previous](#)[Next >](#)

Nullable Type in C#

As you know, a value type cannot be assigned a null value. For example, *int i = null* will give you a compile time error.

C# 2.0 introduced nullable types that allow you to assign null to value type variables. You can declare nullable types using `Nullable<t>` where T is a type.

Example: Nullable type

```
Nullable<int> i = null;
```

A nullable type can represent the correct range of values for its underlying value type, plus an additional *null* value. For example, `Nullable<int>` can be assigned any value from -2147483648 to 2147483647, or a null value.

The Nullable types are instances of `System.Nullable<T>` struct. Think it as something like the following structure.

Example: Nullable struct

```
[Serializable]
public struct Nullable<T> where T : struct
{
    public bool HasValue { get; }
```

```
public T Value { get; }

// other implementation
}
```

A nullable of type *int* is the same as an ordinary *int* plus a flag that says whether the *int* has a value or not (is null or not). All the rest is compiler magic that treats "null" as a valid value.

Example: HasValue

```
static void Main(string[] args)
{
    Nullable<int> i = null;

    if (i.HasValue)
        Console.WriteLine(i.Value); // or Console.WriteLine(i)
    else
        Console.WriteLine("Null");
}
```

Try it


Output:

Null

The `HasValue` returns **true** if the object has been assigned a value; if it has not been assigned any value or has been assigned a null value, it will return **false**.

Accessing the value using `NullableType.Value` will throw a runtime exception if nullable type is null or not assigned any value. For example, `i.Value` will throw an exception if `i` is null:

```
static void Main(string[] args)
{
    Nullable<int> i = null;
    Console.WriteLine(i.Value);
}
```

 `InvalidOperationException – run time`

Invalid use of Nullable Type

Use the `GetValueOrDefault()` method to get an actual value if it is not null and the default value if it is null. For example:

Example: `GetValueOrDefault()`

```
static void Main(string[] args)
{
    Nullable<int> i = null;

    Console.WriteLine(i.GetValueOrDefault());
}
```

Try it

Shorthand Syntax for Nullable Types

You can use the '?' operator to shorthand the syntax e.g. `int?`, `long?` instead of using `Nullable<T>`.

Example: Shorthand syntax for Nullable types

```
int? i = null;  
double? D = null;
```

?? Operator

Use the '??' operator to assign a nullable type to a non-nullable type.

Example: ?? operator with Nullable Type

```
int? i = null;  
  
int j = i ?? 0;  
  
Console.WriteLine(j);
```

Try it

Output:

0


In the above example, i is a nullable int and if you assign it to the non-nullable int j then it will throw a runtime exception if i is null. So to mitigate the risk of an exception, we have used the '??' operator to specify that if i is null then assign 0 to j.

Assignment Rules

A nullable type has the same assignment rules as a value type. It must be assigned a value before using it if nullable types are declared in a function as local variables. If it is a field of any class then it will have a null value by default.

For example, the following nullable of int type is declared and used without assigning any value. The compiler will give **"Use of unassigned local variable 'i'"** error:

```
static void Main(string[] args)
{
    Nullable<int> i;
    Console.WriteLine(i);
}
```



Unassigned nullable type-error

In the following example, a nullable of int type is a field of the class, so it will not give any error.

Example: Nullable type as Class Field

```
class MyClass
{
    public Nullable<int> i;
}
class Program
{
    static void Main(string[] args)
    {
        MyClass mycls = new MyClass();

        if(mycls.i == null)
            Console.WriteLine("Null");
    }
}
```

```
}  
}
```

Try it

Output:

```
Null
```

Nullable Helper Class

Null is considered to be less than any value. So comparison operators won't work against null. Consider the following example where i is neither less than j, greater than j nor equal to j:

Example: Nullable Type Comparison

```
static void Main(string[] args)  
{  
    int? i = null;  
    int j = 10;  
  
    if (i < j)  
        Console.WriteLine("i < j");  
    else if( i > 10)  
        Console.WriteLine("i > j");  
    else if( i == 10)  
        Console.WriteLine("i == j");  
    else
```

```
        Console.WriteLine("Could not compare");  
    }
```

Try it

Output:

```
Could not compare
```

Nullable static class is a helper class for Nullable types. It provides a compare method to compare nullable types. It also has a GetUnderlyingType method that returns the underlying type argument of nullable types.

Example: Helper Class

```
static void Main(string[] args)  
{  
    int? i = null;  
    int j = 10;  
  
    if (Nullable.Compare<int>(i, j) < 0)  
        Console.WriteLine("i < j");  
    else if (Nullable.Compare<int>(i, j) > 0)  
        Console.WriteLine("i > j");  
    else  
        Console.WriteLine("i = j");  
}
```

Try it

Output:

```
i < j
```

Characteristics of Nullable Types

1. Nullable types can only be used with value types.
2. The Value property will throw an InvalidOperationException if value is null; otherwise it will return the value.
3. The HasValue property returns true if the variable contains a value, or false if it is null.
4. You can only use == and != operators with a nullable type. For other comparison use the Nullable static class.
5. Nested nullable types are not allowed. `Nullable<Nullable<int>> i;` will give a compile time error.



Points to Remember :

- 1) `Nullable<T>` type allows assignment of null to value types.
- 2) `?` operator is a shorthand syntax for Nullable types.
- 3) Use value property to get the value of nullable type.
- 4) Use *HasValue* property to check whether value is assigned to *nullable type* or not.
- 5) *Static Nullable* class is a helper class to compare nullable types.



Share



Tweet



Share



Whatsapp

[< Previous](#)[Next >](#)

TUTORIALSTEACHER.COM

TutorialsTeacher.com is optimized for learning web technologies step by step. Examples might be simplified to improve reading and basic understanding. While using this site, you agree to have read and accepted our terms of use and privacy policy.

✉ feedback@tutorialsteacher.com

TUTORIALS

- › ASP.NET Core
- › ASP.NET MVC
- › IoC
- › Web API
- › C#
- › LINQ
- › Entity Framework
- › AngularJS 1
- › Node.js
- › D3.js
- › JavaScript
- › jQuery
- › Sass
- › Https

E-MAIL LIST

Subscribe to TutorialsTeacher email list and get latest updates, tips & tricks on C#, .Net, JavaScript, jQuery, AngularJS, Node.js to your inbox.

Email address

GO

We respect your privacy.

[HOME](#) [PRIVACY POLICY](#) [TERMS OF USE](#) [ADVERTISE WITH US](#)

© 2019 TutorialTeacher.com. All Rights Reserved.