How To Test if Type is Primitive

Asked 9 years, 6 months ago Active 1 year, 2 months ago Viewed 73k times

151

43

I have a block of code that serializes a type into a Html tag.

```
Type t = typeof(T); // I pass <T> in as a paramter, where myObj is of type T
tagBuilder.Attributes.Add("class", t.Name);
foreach (PropertyInfo prop in t.GetProperties())
{
    object propValue = prop.GetValue(myObj, null);
    string stringValue = propValue != null ? propValue.ToString() : String.Empty;
    tagBuilder.Attributes.Add(prop.Name, stringValue);
}
```

This works great, except I want it to only do this for primitive types, like `int` , `double` , `bool` etc, and other types that aren't primitive but can be serialized easily like `string` . I want it to ignore everything else like Lists & other custom types.

Can anyone suggest how I do this? Or do I need to specify the types I want to allow somewhere and switch on the property's type to see if it's allowed? That's a little messy, so it'd be nice if I there was a tidier way.

c# reflection primitive-types

edited Mar 14 '10 at 15:38 asked Mar 14 '10 at 14:58

DaveDev
19.3k 61 179 346

10 `System.String` is not a primitive type. – SLaks Mar 14 '10 at 15:02

2 The better way to do it is to not use generics at all. If you support a small number of types as legal parameter types then simply have that many overloads. If you support any type that implements ISerializable, then write a non-generic method that takes an ISerializable. Use generics for things which are actually *generic*; if the type actually matters, its probably not generic. – Eric Lippert Mar 14 '10 at 20:37

'10 at 5:45

12 Answers

170

You can use the property `Type.IsPrimitive` , but be carefull because there are some types that we can think that are primitives, but they aren´t, for example `Decimal` and `String` .

Edit 1: *Added sample code*

Here is a sample code:

```
if (t.IsPrimitive || t == typeof(Decimal) || t == typeof(String) || ... )
{
    // Is Primitive, or Decimal, or String
}
```

Edit 2: As @SLaks comments, there are other types that maybe you want to treat as primitives, too. I think that you´ll have to add this variations *one by one*.

Edit 3: IsPrimitive = (Boolean, Byte, SByte, Int16, UInt16, Int32, UInt32, Int64, UInt64, IntPtr, UIntPtr, Char, Double, and Single), Anther Primitive-Like type to check (t == typeof(DateTime))

edited May 23 '17 at 11:55
Community ♦
1 1

answered Mar 14 '10 at 15:02
Javier
3,452 2 16 18

10 And perhaps `DateTime` , `TimeSpan` , and `DateTimeOffset` . – SLaks Mar 14 '10 at 15:08

2 You need to use the logical or (`||`), not the bitwise or (`|`). – SLaks Mar 14 '10 at 15:14

1 Why the bitwise or? – Motti Mar 14 '10 at 15:14

36 Here's an extension method I wrote to conveniently run the tests described in the answers by @Javier and Michael Petito: gist.github.com/3330614. –
Jonathan Aug 12 '12 at 8:13

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email Sign up with Google Sign up with Facebook

51

I just found this question while looking for a similar solution, and thought you might be interested in the following approach using `System.TypeCode` and `System.Convert`.

It is easy to serialize any type that is mapped to a `System.TypeCode` other than `System.TypeCode.Object`, so you could do:

```
object PropertyValue = ...
if(Convert.GetTypeCode(PropertyValue) != TypeCode.Object)
{
    string StringValue = Convert.ToString(PropertyValue);
    ...
}
```

The advantage with this approach is you don't have to name every other acceptable non-primitive type. You could also modify the above code slightly to handle any type that implements IConvertible.

answered Apr 18 '10 at 23:31
Michael Petito
10.8k 2 32 48

2 This is great, I had to manually add `Guid` for my own purposes (as a primitive in my definition). – Erik Philips Jul 27 '12 at 22:54

46

We do it like this in our ORM:

```
Type t;
bool isPrimitiveType = t.IsPrimitive || t.IsValueType || (t == typeof(string));
```

I know that using `IsValueType` is not the best option (you can have your own very complex structs) but it works in 99% cases (and includes Nullables).

edited Jul 17 '18 at 2:17
MGOwen
3,168 10 46 62

answered May 14 '12 at 10:45
Alex
30.5k 12 93 88

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email Sign up with Google Sign up with Facebook

2 @xhafan: You answer the wrong question. All structs are like `decimal` in that regard. But are there a type for which `IsPrimitive` returns `true` but `IsValueType` returns `false` ? If there are no such type then the `t.IsPrimitive` test is unnecessary. – Lii Jan 13 '15 at 8:21

4 @Lii you are right, every primitive type has `IsValueType` set to true, so checking for `IsPrimitive` is not need. Cheers! – xhafan Jan 13 '15 at 14:17

So what is then the answer to "why does the framework provide 2 properties that represent the very same data" ? – Veverke Apr 20 '15 at 14:33

29

From @Ronnie Overby response and @jonathanconway comment, I wrote this method that works for Nullable, and exclude user structs.

```
public static bool IsSimpleType(Type type)
{
    return
        type.IsPrimitive ||
        new Type[] {
            typeof(Enum),
            typeof(String),
            typeof(Decimal),
            typeof(DateTime),
            typeof(DateTimeOffset),
            typeof(TimeSpan),
            typeof(Guid)
        }.Contains(type) ||
        Convert.GetTypeCode(type) != TypeCode.Object ||
        (type.IsGenericType && type.GetGenericTypeDefinition() == typeof(Nullable<>) &&
IsSimpleType(type.GetGenericArguments()[0]))
        ;
}
```

With the following TestCase :

```
struct TestStruct
{
    public string Prop1;
    public int Prop2;
}
```

```
        [Test]
        public void Test1()
        {
            Assert.IsTrue(IsSimpleType(typeof(Enum)));
            Assert.IsTrue(IsSimpleType(typeof(String)));
            Assert.IsTrue(IsSimpleType(typeof(Char)));
            Assert.IsTrue(IsSimpleType(typeof(Guid)));

            Assert.IsTrue(IsSimpleType(typeof(Boolean)));
            Assert.IsTrue(IsSimpleType(typeof(Byte)));
            Assert.IsTrue(IsSimpleType(typeof(Int16)));
            Assert.IsTrue(IsSimpleType(typeof(Int32)));
            Assert.IsTrue(IsSimpleType(typeof(Int64)));
            Assert.IsTrue(IsSimpleType(typeof(Single)));
            Assert.IsTrue(IsSimpleType(typeof(Double)));
            Assert.IsTrue(IsSimpleType(typeof(Decimal)));

            Assert.IsTrue(IsSimpleType(typeof(SByte)));
            Assert.IsTrue(IsSimpleType(typeof(UInt16)));
            Assert.IsTrue(IsSimpleType(typeof(UInt32)));
            Assert.IsTrue(IsSimpleType(typeof(UInt64)));

            Assert.IsTrue(IsSimpleType(typeof(DateTime)));
            Assert.IsTrue(IsSimpleType(typeof(DateTimeOffset)));
            Assert.IsTrue(IsSimpleType(typeof(TimeSpan)));

            Assert.IsFalse(IsSimpleType(typeof(TestStruct)));
            Assert.IsFalse(IsSimpleType(typeof(TestClass1)));

            Assert.IsTrue(IsSimpleType(typeof(Nullable<Char>)));
            Assert.IsTrue(IsSimpleType(typeof(Nullable<Guid>)));

            Assert.IsTrue(IsSimpleType(typeof(Nullable<Boolean>)));
            Assert.IsTrue(IsSimpleType(typeof(Nullable<Byte>)));
            Assert.IsTrue(IsSimpleType(typeof(Nullable<Int16>)));
            Assert.IsTrue(IsSimpleType(typeof(Nullable<Int32>)));
            Assert.IsTrue(IsSimpleType(typeof(Nullable<Int64>)));
            Assert.IsTrue(IsSimpleType(typeof(Nullable<Single>)));
            Assert.IsTrue(IsSimpleType(typeof(Nullable<Double>)));
            Assert.IsTrue(IsSimpleType(typeof(Nullable<Decimal>)));

            Assert.IsTrue(IsSimpleType(typeof(Nullable<SByte>)));
            Assert.IsTrue(IsSimpleType(typeof(Nullable<TimeSpan>)));

            Assert.IsFalse(IsSimpleType(typeof(Nullable<TestStruct>)));
        }
```

answered Sep 1 '15 at 17:47
Xav987
702 8 16

15

Here's how I did it.

```
    static class PrimitiveTypes
    {
        public static readonly Type[] List;

        static PrimitiveTypes()
        {
            var types = new[]
                          {
                              typeof (Enum),
                              typeof (String),
                              typeof (Char),
                              typeof (Guid),

                              typeof (Boolean),
                              typeof (Byte),
                              typeof (Int16),
                              typeof (Int32),
                              typeof (Int64),
                              typeof (Single),
                              typeof (Double),
                              typeof (Decimal),

                              typeof (SByte),
                              typeof (UInt16),
                              typeof (UInt32),
                              typeof (UInt64),
           var nullTypes = from t in types
                           where t.IsValueType
                           select typeof (Nullable<>).MakeGenericType(t);

           List = types.Concat(nullTypes).ToArray();
       }

       public static bool Test(Type type)
       {
           if (List.Any(x => x.IsAssignableFrom(type)))
               return true;

           var nut = Nullable.GetUnderlyingType(type);
           return nut != null && nut.IsEnum;
       }
   }
```

edited Apr 6 '15 at 20:50

answered Mar 22 '13 at 19:08
Ronnie Overby
24.9k 63 232 327

I would add Guid as well, but it is a pretty good list – MeTitus Jan 23 '15 at 20:10

2 Nice work on that. – Jerry Nixon Apr 10 '15 at 23:16

@RonnieOverby. is there any paticular reason you use `IsAssignableFrom` in your test instead of contains? – johnny 5 Mar 19 '18 at 4:10

5

Also a good possibility:

```
private static bool IsPrimitiveType(Type type)
{
    return (type == typeof(object) || Type.GetTypeCode(type) != TypeCode.Object);
}
```

edited Jun 13 '17 at 9:50

answered Jun 14 '13 at 13:16

3 Neither `String` nor `Decimal` are primitives. – k3flo Jun 19 '13 at 18:20

This works for me, but I renamed to IsClrType to not confuse its meaning with the existing .IsPrimitive on the Type class – KnarfaLingus Apr 28 '15 at 21:28

This won't pick Guid or TimeSpan, for example. – Stanislav Jun 13 '17 at 9:21

3

Assuming you have a function signature like this:

```
void foo<T>()
```

You could add a generic constraint to allow value types only:

```
void foo<T>() where T : struct
```

Notice that this allows not only primitive types for T, but any value type.

answered Mar 14 '10 at 15:04
eWolf
2,179 2 24 40

2

I had a need to serialize types for the purposes of exporting them to XML. To do this, I iterated through the object and opted for fields that were primitive, enum, value types or serializable. This was the result of my query:

```
Type contextType = context.GetType();

var props = (from property in contextType.GetProperties()
                         let name = property.Name
                         let type = property.PropertyType
                         let value = property.GetValue(context,
                                     (BindingFlags.GetProperty | BindingFlags.GetField |
BindingFlags.Public),
```

Join Stack Overflow to learn, share knowledge, and build your career.

Sign up with email Sign up with Google Sign up with Facebook

I used LINQ to iterate through the types, then get their name and value to store in a symbol table. The key is in the 'where' clause that I chose for reflection. I chose primitive, enumerated, value types and serializable types. This allowed for strings and DateTime objects to come through as I expected.

Cheers!

answered May 18 '13 at 14:55

JFalcon
107 1 7

1

This is what I have in my library. Comments are welcome.

I check IsValueType first, since it handles most types, then String, since it's the second most common. I can't think of a primitive that isn't a value type, so I don't know if that leg of the if ever gets hit.

```
  Public Shared Function IsPersistable(Type As System.Type) As Boolean
    With TypeInformation.UnderlyingType(Type)
      Return .IsValueType OrElse Type = GetType(String) OrElse .IsPrimitive
    End With
  End Function

  Public Shared Function IsNullable(ByVal Type As System.Type) As Boolean
    Return (Type.IsGenericType) AndAlso (Type.GetGenericTypeDefinition() Is
GetType(Nullable(Of )))
  End Function

  Public Shared Function UnderlyingType(ByVal Type As System.Type) As System.Type
    If IsNullable(Type) Then
      Return Nullable.GetUnderlyingType(Type)
    Else
      Return Type
    End If
  End Function
```

Then I can use it like this:

```
                       Select PropertyInfo
    End Function
```

answered May 22 '12 at 16:38
toddmo
10.7k 8 69 80

1

I just want to share my solution. Perhaps it's useful to anyone.

```
public static bool IsPrimitiveType(Type fieldType)
{
   return fieldType.IsPrimitive || fieldType.Namespace.Equals("System");
}
```

answered Aug 27 '13 at 8:07
Bahamut
187 1 6

5 `IsPrimitiveType(typeof(System.AccessViolationException)) == true` – Ronnie Overby Apr 6 '15 at 20:48

2 `namespace System { class MyNonPrimitiveType { } }` – Ronnie Overby Apr 6 '15 at 20:48

0

```
public static bool IsPrimitiveType(object myObject)
{
   var myType = myObject.GetType();
   return myType.IsPrimitive || myType.Namespace == null ||
myType.Namespace.Equals("System");
}
```

Don't forget to check NULL namespace, because anonymous objects don't have assigned namespace

Isn't this type going to be "object" always? From the parameter... – Andre Jul 23 at 19:30

0

Here is another viable option.

```
public static bool CanDirectlyCompare(Type type)
{
    return typeof(IComparable).IsAssignableFrom(type) || type.IsPrimitive ||
type.IsValueType;
}
```

answered Dec 9 '15 at 22:39

user2023116

115 1 2 13