

Re: String Reference Type

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.framework.aspnet/2008-02/msg00997.html>

- *From:* "Lars" <jon.doe@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Wed, 13 Feb 2008 00:30:49 GMT
-

Hi

Check out the page

[http://msdn2.microsoft.com/en-us/library/aa691324\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/aa691324(VS.71).aspx)

Out take from the page

C# Language Specification

7.2.2 Operator overloading

All unary and binary operators have predefined implementations that are automatically available in any expression. In addition to the predefined implementations, user-defined implementations can be introduced by including operator declarations in classes and structs (Section 10.9). User-defined operator implementations always take precedence over predefined operator implementations: Only when no applicable user-defined operator implementations exist will the predefined operator implementations be considered.

Does this mean that you can override all operators except assignment = operator.

Lars

"Anthony Jones" <Ant@xxxxxxxxxxxxxxxxxxxx> skrev i meddelandet news:uHjEmzcbIHA.1168@xxxxxxxxxxxxxxxxxxxxxxxxxxxx

"Lars" <jon.doe@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message [news:jMisj.3767\\$R_4.2820@xxxxxxxxxxxxxxxxxxxxxxxxxxxx](mailto:news:jMisj.3767$R_4.2820@xxxxxxxxxxxxxxxxxxxxxxxxxxxx)

Hi

Re: String Reference Type

So to the original point; is a string a class or a structure?

Ans:

It's

a
class.
Therefore its always allocated in the heap and a variable of
string
type
is
always a reference to this string object allocated on the heap.

```
Dim x As String = "Hello World"  
Dim y As String  
y = x
```

As with all classes in this case y and x both reference the
same String
object.

Does the String class implements operators to take care of this.

Can you write you own class for example Complex that defines the

assignment

operator and comparement operators. Si that if you wrote Z1=Z2 copies the
values Z2.x and Z2.y to Z1 leavind Z2 unchained. Do you have to create a

new

object of Z and return for this?

Neither VB nor C# support the overloading of assignment operators or the
concept of a copy constructor.

If your class is simple enough such that all of its fields are structures
(as above) then you can make a copy of class with the inherited
.MemberwiseClone method that comes from object:–

```
//C#  
Z1 = (Complex)Z2.MemberwiseClone()
```

```
'VB  
Z1 = DirectCast(Z2.MemberwiseClone(), Complex)
```

Re: String Reference Type

For more complex objects where some fields are classes memberwise clone would not clone the referenced classes. For that you would need to implement ICloneable and implement the Clone method to perform the Deep copy and the referenced classes themselves either need to be simple enough that a memberwiseclone would suffice or also implement ICloneable.

A structure would seem to be a better basis for a complex number. Using a structure the $Z1 = Z2$ would by nature create a copy of the member fields. The mathematical and comparison operators can then be overloaded to provide the appropriate behaviour.

One thing about parameter passing that might confuse VB6ers is that Structures can be passed ByVal where they couldn't in VB6. Therefore using a structure for small composite data structures is actually quite viable in VB.NET.

An alternative would be to use an immutable class. There are pros and cons to both approaches.

What makes this less obvious (as Mykola alluded to) is the immutable nature of a String. Once a string object has been allocated and initialised its content cannot be changed, not even changes that don't involve changing the length. This results in code which makes strings look like they are behaving as structures because no where do see an operation on one reference that is visible through another reference.

A similar type to a string is a character array but a character array is mutable :-

```
Dim z As Char() = New Char(2) {"A"c, "B"c, "C"c}  
Dim x As Integer()
```

Re: String Reference Type

```
x = z
```

```
x(1) = "X"c
```

```
Console.WriteLine(z(1))
```

Note that both z and x reference the same Array object. A modification made to the object via one reference is visible using the other. This sort

of

operation just isn't available on the string object. Its the lack of

this

sort of thing happening in code to a string which gives the illusion of

it

being a structure.

However Mykola is in error stating that you can't pass strings as ByRef. You can (in both VB and C#).

```
Public Sub DoSomething(ByRef rs As String)  
rs = "New content"  
End Sub
```

```
Dim s as String = "Hello World"
```

```
DoSomething(s)
```

```
Console.WriteLine(s)
```

This confusion arises from the overloaded use of the word Reference

(which

is why I prefer to think in terms of Structures or Classes). When you

Re: String Reference Type

think about it all variables are ultimately references to somewhere in memory where a value is stored. For example:–

```
Sub Thing()  
Dim i as Integer  
Dim o as Object
```

The symbol i resolves to a offset pointer from the a fixed point on the stack frame where 4 bytes hold the actual value of the integer. The symbol o resolves to an offset pointer (likely the same as for i + 4) from the same fixed point on the stack frame which will hold a reference to some reference to an object stored on the heap.

Passing ByRef has nothing to do with whether the value is a Structure or

a

Class but everything to do with where relative to a fixed point on a

stack

frame to find the variable value (be that the actual structure content

or

the reference to a heap held object). Since at the time of call the

size

of the current frame is known its relatively simple to supply an offset

from

a fixed point in the new frame back into the previous.

Re: String Reference Type

I Think I got it

```
public void swap( ref String S1, ref String S2 )
{
String tmp1=S1;
S2=S1;
S1=tmp;
}
```

Yes that'll work. Not only that but no new string is ever allocated. In VB6 such an operation resulted in 3 string allocation/dealloactions. (Although hardcore VBers wouldn't do such a swap in this way).

What this function does is swapping the reference values. Compare it to the

following C++ and Delphi code

```
public void swap( String&* S1, String&* S2 )
{
String* tmp1=S1;
S2=S1;
S1=tmp;
}
```

Yes this is an Identical operation.

--
Anthony Jones – MVP ASP/ASP.NET

Note 1:-

A couple of other reasons (apart from my primary reason that the word reference is already over used) I prefer to think in terms of Structure

or

Re: String Reference Type

Class instead of ValueType or ReferenceType are:-

When you take a basic course in Datastructures and Algorithms you learn than

a

REFERENCE is a reference that refers to another object. Whether it's implemented by pointers, memory on the heap or any thing else doesn't

matter.

A Value is value treated as a value (object). The OBJECT if you prefer. Every Reference has to refer to an OBJECT. A teacher I had at the

University

(Basic Datatypes and Algorithms) told us to look at References is a non language dependant matter as below. 14 years later I still recall it.

Yeah but I find all the Computer science stuff abstracts things more than most humans are comfortable with. There are lots of things which can be considered references but if we kept calling them all 'reference' then we wouldn't know which of the various possible 'implementations' of reference we're talking about.

References (pointers for old C programmers, instances of classes in C#)
{Reference} ----> {Object (1,2,3,4,5,6,67,78,)}

or

{Reference} ----> {Object (1,2,3,4,5,6,67,78,),Object (1,2,3,4,5,6,67,78),Object("EHROIERO") }

Objects

{Object (1,2,3,4,5,6,67,78,)}

{Object (1,2,3,4,5,6,78,)}

{Object ("DJÖLFD","DODHFO")}

1). Its possible to have a reference to a structure. I.e. when its

boxed

Re: String Reference Type

for example referenced via variable typed as Object.

2.) The MustInherit (abstract) type System.ValueType from which all 'ValueTypes' inherit is a class not a structure. How can a structure inherit from a Class? Ans: it can't. Only when Boxed does a Structure inherit from ValueType and at the point its a reference to copy on the heap.

For those of you who have hard understanding heap and stack. Look at the example above. A picture says more than a thousand words as they say. If

the

object referred to is on the heap, file, on another server doesn't matter.
Think abstract and you get it.

Yes there is an abstraction to be had here but whilst abstraction is a good developers bread and butter its possible to over do it.

Its all quite confusing.

Not really wonce you know that all instances of classes are references it's no problem.

I don't think you are speaking for the majority there though.

What I did was to not only mix C++ and C# I also have a lot of Delphi code in memmory. In the matter of how to implement instances of classes Delphi is more like C# than C++ is. In a true Obecj Orienter Language as Smalltalk every thing is a reference. As I understand C# is

move

Re: String Reference Type

towards that direction. But C# is not near an truly Object Oriented language. Although it you can use OOP techniques. The build in classes
oin
C# doesn't support encapsulation fully. For example

```
// Does LabekX know that you change the value of the Text when I do  
this?  
Does C# allow you to define properties like Delphi.  
LabelX.Text = "Some text";
```

```
private string myString  
string text  
{  
get { return myString; }  
set { myString = value; }  
}
```

I think you might need to read up C#' more before making statements like 'not near a truly object oriented language'. Besides where in the OO book does it say that a true object supports properties anyway. In fact properties as singe construct to unify what were otherwise known as Accessor and Mutator methods is a relatively recent concept in OO languages. I've even heard C++ zealots slate VB6 for supporting them.

This is important to think of when you wrote your own classes. For every member (variable) in the class write an accessor and modifier method. In general Set Get methods. It makes life easy when you inherit the class in another class.

I certainly agree with this. In fact I would take it further and suggest that in many cases internal code the is using these private fields should actually use the property methods instead. This allows overrides of such members to be used by such code.

The terms ValueType and ReferenceType apply globally to
an Object
(which
is

Re: String Reference Type

always a reference) and indicates whether the type is 'normally' stored directly in the variable declared of that type or whether the variable

of

that type always holds a reference to the value held on the heap.

--

Anthony Jones – MVP ASP/ASP.NET