VB : Types & Assemblies

Interfacing with VB

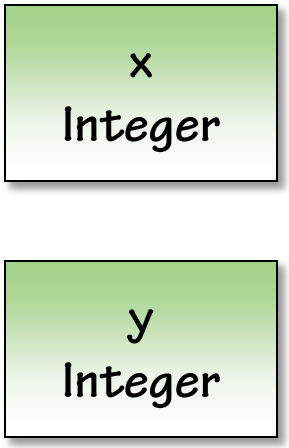
Overview

- Value types and reference types
- Parameter passing
- Type conversions
- Assemblies, references and namespaces

Value Types

- **Values allocated on the Stack**
  - Memory released when variable goes out of scope
- **Many built-in primitives are value types**
  - All numeric data types, boolean, date, enumerations
- **New operator is not used when creating a value type**
- **Assignment copies the *value* of the variable**

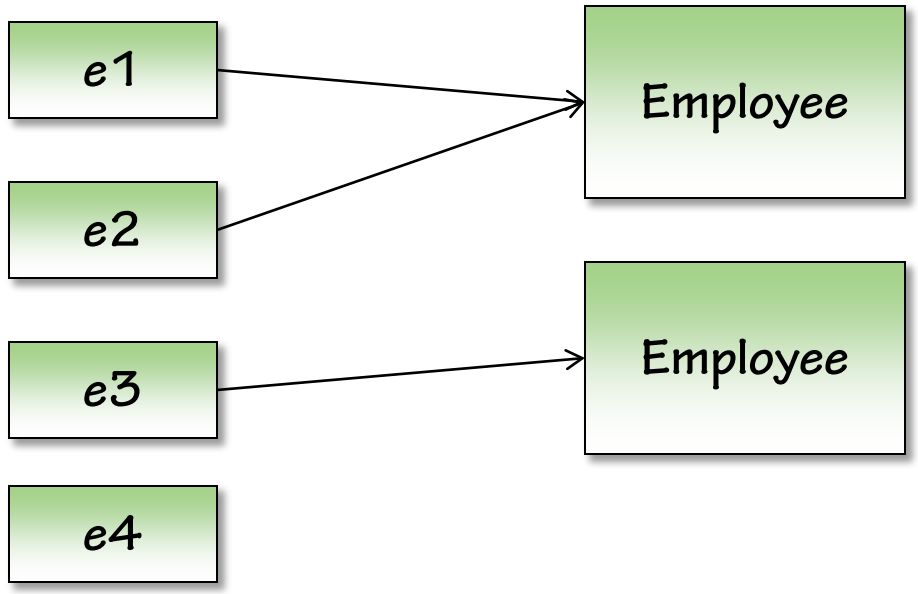
```
Dim x As Integer = 5
Dim y As Integer
y = x
x = 10
Console.WriteLine(x) ' 10
Console.WriteLine(y) ' 5
```



Reference Types

- Objects are created using the New operator
- Variables store a reference to an object
- Reference is stored on the Stack, object is stored on the Heap
- Assignment copies the reference
- Multiple variables can point to the same object
- Variables may not have a reference (be set to Nothing)

```
Dim e1 As New Employee()
Dim e2 As Employee = e1
Dim e3 As New Employee()
Dim e4 As Employee = _
    Nothing
```



The Magical String Type

- **Strings are reference types**
  - But behave like value types
  - Immutable
    - Changing the value of a variable creates a new String object that replaces the old one

```
Dim s1 As String = "Scott"
Dim s2 As String = s1

s2 = "Rob"
Dim s3 As String

Console.WriteLine(s1)
Console.WriteLine(s2)
Console.WriteLine(IIf(s3 is Nothing, "Nothing", s3))
```

Parameter Passing

- **Passing parameters by value (ByVal)**
  - Reference types pass a copy of the reference
  - Value types pass a copy of the value
    - Changes to value don't propagate to caller
- **Passing parameters by reference (ByRef)**
  - Changes to the parameter value are propagated to the caller

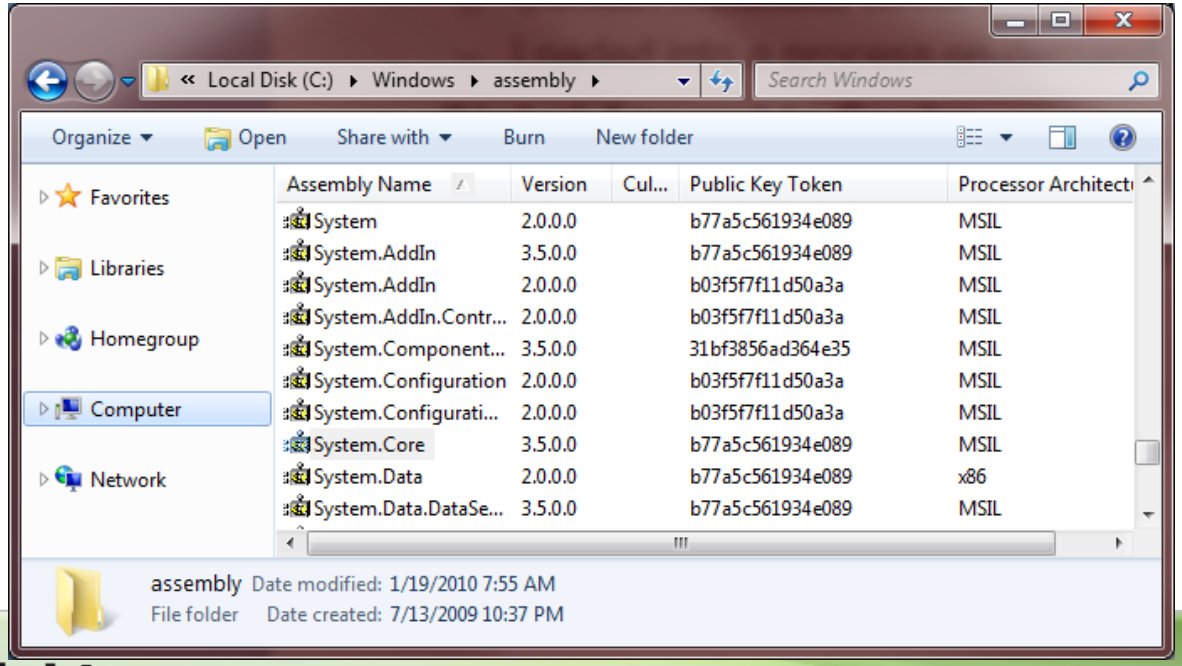
```
Public Function DoWork(ByVal code As String, _
    ByRef units As Integer) As Boolean

    ' function implementation
End Function
```

Type Conversion and Option Strict

- **Sometimes you need to convert a value from one type to another**
  - Examples: Convert “100” or 3.14 to an Integer
- **By default, Visual Basic will do these conversions for you**
  - This can sometimes lead to data loss or unexpected results
- **You can set Option Strict On to limit automatic type conversions**
  - Option Strict can be set at the file or project level
  - Widening conversions (those that will not result in data loss) remain automatic
    - Example 100 can be converted to Double but 3.14 could not be converted to Integer
  - For other conversions, you need to use conversion functions
- **Conversion functions:**
  - Native Visual Basic: CInt(), CBool(), CDbl(), CType()
  - .NET Framework: Convert.ToInt32(), Convert.ToBoolean(), Convert.ToDouble()
  - String Parsing: Integer.Parse(), Boolean.Parse(), Double.Parse()

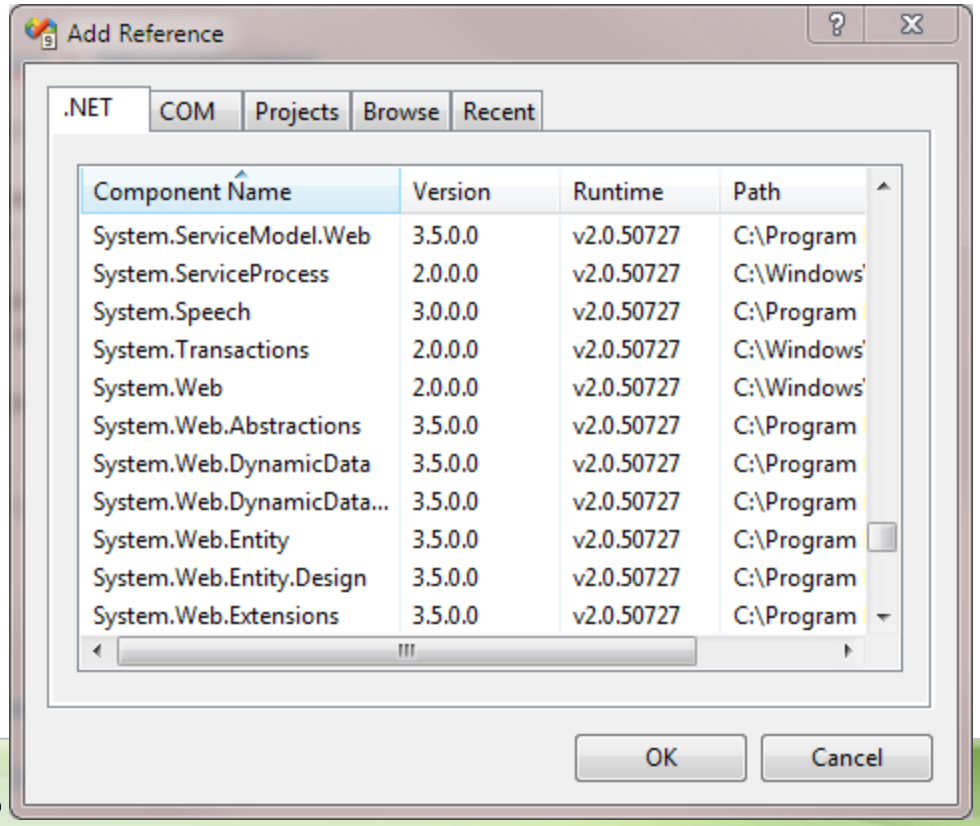
Assemblies

- **Fundamental building blocks**
  - Implemented as .exe or .dll files
  - Contain metadata about version and all types inside
- **Global Assembly Cache**
  - A central location to store assemblies for a machine



References

- **Must load assembly into a process before using types inside**
  - Easy approach – reference the assembly in Visual Studio
  - Assemblies loaded on demand at runtime



Namespaces

- **Namespace organize types**
  - Avoid type name collisions
  - Can define namespace in one or more places
- **Fully qualified type names**
  - Includes the assembly name
  - Includes the namespace
  - Includes the type name
- **Imports**
  - Brings other namespaces into scope
  - No need to namespace qualify a Type

```
Imports System
Imports System.Net
Imports System.Data
Imports System.Linq
Imports System.Text
```

Summary

- Value types and reference types
- Parameter passing
- Type conversions
- Assemblies, references and namespaces