CONVERTING DATA TYPES

IMPLICIT TYPE CONVERSION

When you assign a value to one date type to a variable of another data type, Visual Basic attempts to convert the value being assigned to the data type of the receiving variable. This is known as **implicit type conversion**. For example, let's say we want to assign the integer 5 to a variable of type **Single** named **num**:

```
Dim num As Single = 5
```

When the statement executes, the integer 5 is automatically converted into the real number 5.0, which is then stored in the variable num. This conversion is what's called a **widening conversion** because no data is lost.

NARROWING CONVERSIONS

If you assign a real number to an integer variable, Visual Basic attempts to perform what's called a narrowing conversion. Oftentimes, some data is lost. For example, the following statement assigns 12.2 to an integer variable:

```
Dim num As Integer = 12.2
```

The value 12.2 is rounded down to 12. Similarly, the next statement rounds up to the nearest integer (i.e. 13) when the fractional part of the number is greater than .5:

```
Dim num As Integer = 12.6
```

Another narrowing conversion occurs when assigning a **Double** value to a variable type **Single**. Both hold floating-point values, but **Double** permits more significant digits:

```
Dim numOne As Double = 1.2345678
Dim numTwo As Single = numOne
```

The value stored in **numTwo** is rounded up to 1.234568 because variables of **Single** type can only hold seven significant digits.

CONVERTING STRINGS TO NUMBERS

Under some circumstances, Visual Basic will try to convert string values to numbers. In the following statement, for example, "12.2" is a string containing a numeric expression:

```
Dim num As String = "12.2"
```

The string “12.2” is a string of characters in a numeric-like format that cannot be used for calculations. When the following statements execute, the string “12.2” is converted to the number 12.2:

```
Dim num As Single
num = “12.2”
```

If we assign the string “12.2” to an **Integer** variable, the result is rounded down to 12:

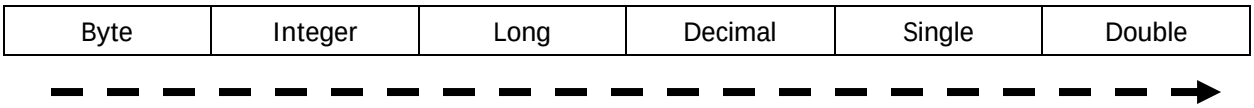
```
Dim num As Integer
num = “12.2”
```

A similar effect occurs when the user enters a number into a **TextBox** control. We usually want to assign the contents of the text box’s **Text** property to a numeric value. The Text property is by definition type String, so its contents are implicitly converted to a number when we assign it to a numeric variable:

```
Dim num As Integer
num = txtValue.Text
```

OPTION STRICT

Visual Basic has a configuration option named **Option Strict** that determines whether certain implicit conversions are legal. If you set **Option Strict** to **ON**, only widening conversions are permitted (such as Integer to Single). The following diagram shows how implicit conversion between numeric types must be in a left-to-right direction:



A Single value can be assigned to a variable of type Double, an Integer can be assigned to a variable of type **Single**, etc.

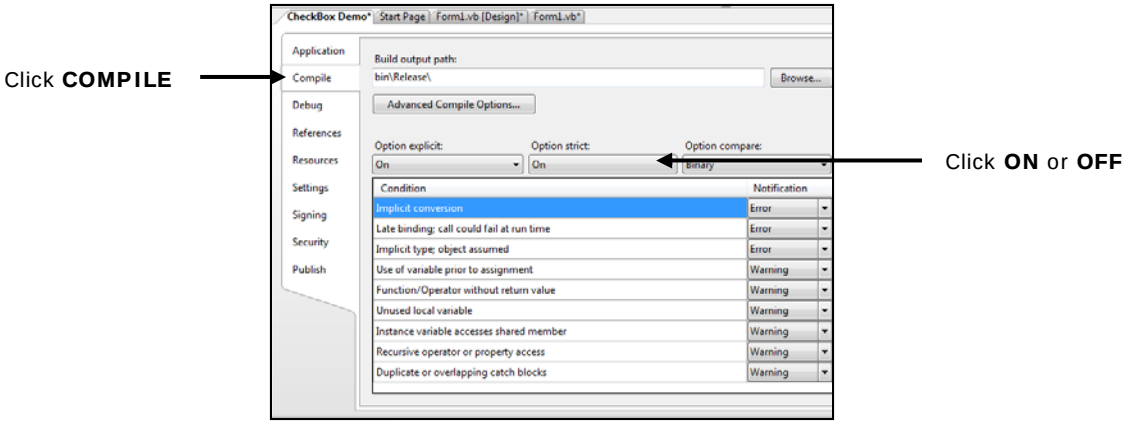
If, on the other hand, **Option Strict** is set to **OFF**, all types of numeric conversions are permitted, with possible loss of data.

Option Strict can be set in two different ways:

1. By inserting an **Option Strict** statement at the top of the source code file:

   ```
   Option Strict On
   ```

2. Right-click the project name in the **Solution Explorer** window, select **Properties**, and then select the **Compile** tab. From the **Option Strict** drop-down list, you can select **ON** or **OFF**.



It is recommended that you use **Option Strict** to catch errors that result when you accidentally assign a value of the wrong type to a variable. When set to **ON**, **Option Strict** forces you to use a conversion function, making your intentions clear, and helps to avoid runtime errors.

EXPLICIT TYPE CONVERSIONS

The following table outlines the most commonly used Visual Basic conversion functions. The input to each of the conversion functions is an expression, which is another name for a variable name, an arithmetic expression, etc.

These conversion functions are required when **Option Strict** is set to **ON** and you need to assign a wider numeric type to a narrower numeric type (e.g. Long to Integer, Double to Single, etc.)

FUNCTION	DESCRIPTION
CChar(*expr*) or Convert.ToChar(expr)	• Converts a string expression to a Char. • If the string contains more than one character, only the first character is returned.
CBool(*expr*) or Convert.ToBoolean(expr)	• Converts a string expression (that must equal **True** or **False**) to a Boolean. • If the expression does not equal **True**, **False** or **Nothing**, a runtime error is generated.
CDbl(*expr*) or Convert.ToDouble(*expr*)	• Converts a numeric or string expression to a Double. • If the expression converts to a value outside the range of a Double, or is not a numeric value, a runtime error is generated.
CInt(*expr*) or Convert.ToInt32(*expr*)	• Converts a numeric or string expression to an Integer. • If the expression converts to a value outside the range of an Integer, or is not a numeric value, a runtime error is generated.
CLng(*expr*) or Convert.ToInt64(expr)	• Converts a numeric or string expression to a Long. • If the expression converts to a value outside the range of a Long, or is not a numeric value, a runtime error is generated.
CSng(*expr*) or Convert.ToSingle(*expr*)	• Converts a numeric or string expression to a Single.

	• If the expression converts to a value outside the range of a Single, or is not a numeric value, a runtime error is generated.
CStr(*expr*) or Convert.ToString(*expr*)	• Converts a numeric, Boolean, Date or String expression to a String.

THE *ToString()* METHOD

Each Visual Basic data type has a **`ToString()`** method, which returns a string representation of the variable calling the method. You call the **`ToString()`** method using the following general format:

```
VariableName.ToString()
```

The following code segment shows an example of the method's use:

```
Dim num As Integer = 123
lblNumber.Text = num.ToString()
```