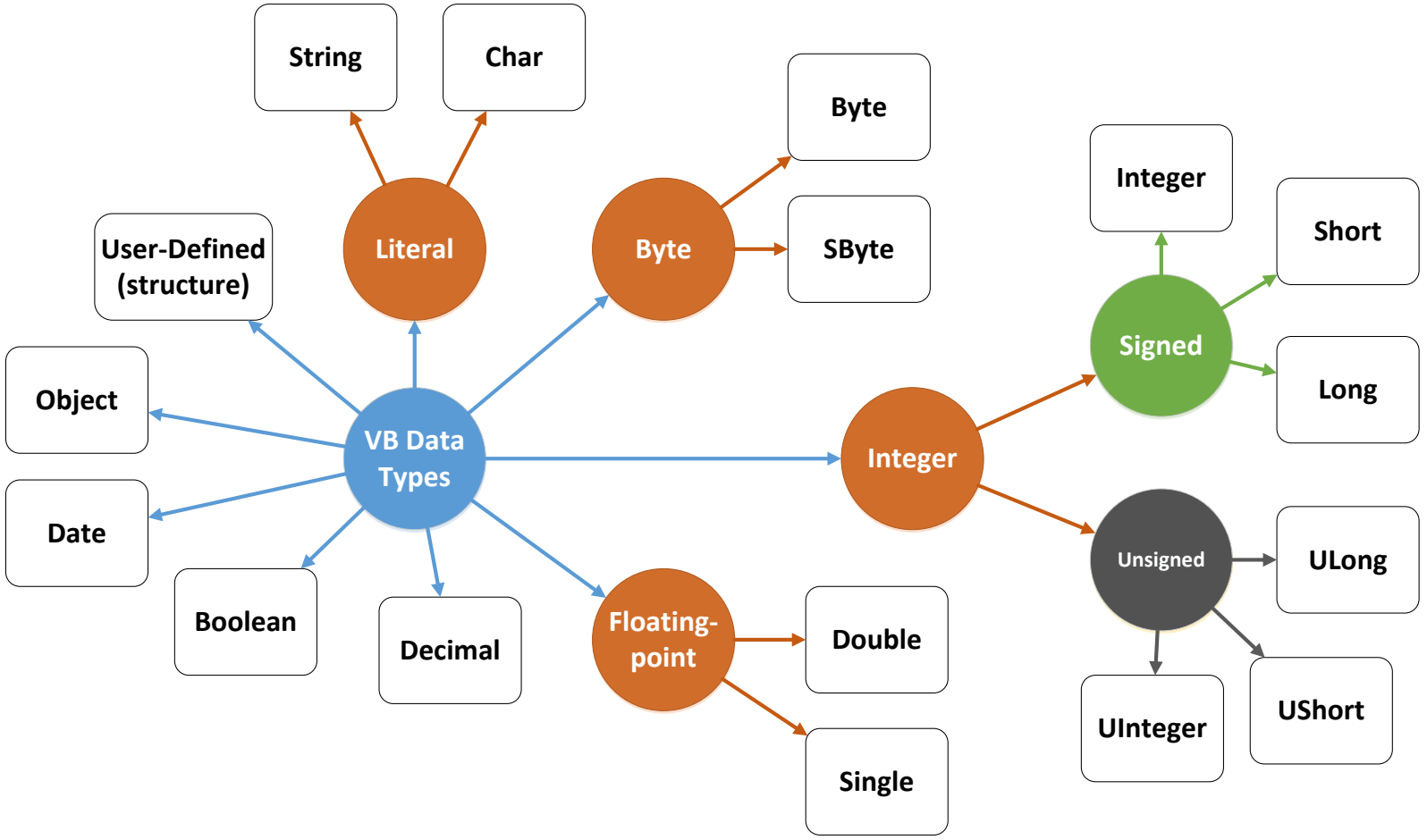
[02] Data Types and Variable

308-333 VISUAL PROGRAMMING AND APPLICATIONS

Data Types


String
Char
Literal
User-Defined (structure)
Object
Date
Boolean
Decimal
Byte
SByte
Byte
Floating-point
Double
Single
VB Data Types
Integer
Integer
Short
Signed
Long
Unsigned
ULong
UInteger
UShort

Data Types

Visual Basic type	CLR type structure	Storage
Boolean	Boolean	1 byte
Byte	Byte	1 byte
Char (single character)	Char	2 bytes
Date	DateTime	8 bytes
Decimal	Decimal	16 bytes
Double (double-precision floating-point)	Double	8 bytes
Integer	Int32	4 bytes
Long (long integer)	Int64	8 bytes
Object	Object (class)	4 bytes (32-bit platform) 8 bytes (64-bit platform)

Data Types

Visual Basic type	CLR type structure	Storage
SByte	SByte	1 byte
Short	Int16	2 bytes
String (variable-length)	String (class)	Depends on implementing platform
UInteger	UInt32	4 bytes
ULong	UInt64	8 bytes
User-Definded (structure)	(inherits from ValueType)	Depends on implementing platform
UShort	UInt16	2 bytes

Data Types

Boolean

Holds values that can be only **True** or **False**.

The keywords True and False correspond to the two states of Boolean variables.

The **default** value of Boolean is **False**.

Boolean values are not stored as numbers, and the stored values are not intended to be equivalent to numbers.

```
Dim runningVB As Boolean
' Check to see if program is running on Visual Basic engine.
If scriptEngine = "VB" Then
    runningVB = True
End If
```

Data Types

Boolean

When Visual Basic converts numeric data type values to Boolean, **0** becomes **False** and all **other values** become **True**.

When Visual Basic converts Boolean values to numeric types, **False** becomes **0** and **True** becomes **-1**.

Data Types

Byte

Holds unsigned 8-bit (1-byte) integers that range in value from 0 through 255.

Use the Byte data type to contain **binary data**.

The **default** value of Byte is **0**.

```
Dim b As Byte
b = 30
' Bit shift to the right divides the number in half. In this
' example, binary 110 becomes 11.
b >>= 1
```

Data Types

Char

Holds unsigned 16-bit (2-byte) code points ranging in value from 0 through 65535. Each code point, or character code, represents a single **Unicode** character.

Use the Char data type when you need to hold only a **single character** and do not need the overhead of String.

In some cases you can use **Char()**, an *array of Char* elements, to hold multiple characters.

The **default** value of Char is the character with a code point of **0**.

```
Dim charVar As Char
charVar = "Z"
```

Data Types

Date

Holds **IEEE 64-bit (8-byte)** values that represent dates.

Ranging from **January 1 of the year 0001** through **December 31 of the year 9999**, and times from **12:00:00 AM (midnight)** through **11:59:59.9999999 PM**.

Each increment represents 100 nanoseconds of elapsed time since the beginning of January 1 of the year 1 in the Gregorian calendar.

The maximum value represents 100 nanoseconds before the beginning of January 1 of the year 10000.

Use the Date data type to contain date values, time values, or date and time values.

The **default** value of Date is **0:00:00 (midnight) on January 1, 0001**.

```
Dim someDate As Date =  #8/13/2002 12:14 PM#
```

Data Types

Date

You must enclose a Date literal within number signs (# #).

You must specify the date value in the format **M/d/yyyy**, for example #5/31/1993#. This requirement is independent of your locale and your computer's date and time format settings.

```
MsgBox("The formatted date is " & Format(#5/31/1993#, "dddd, d MMM yyyy"))
```

Data Types

Decimal

Holds signed 128-bit (16-byte) values representing 96-bit (12-byte) integer numbers scaled by a variable power of 10.

The scaling factor specifies the number of digits to the right of the decimal point; it ranges from 0 through 28.

With a scale of 0 (no decimal places), the largest possible value is +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9228162514264337593543950335E+28).

With 28 decimal places, the largest value is +/-7.9228162514264337593543950335, and the smallest nonzero value is +/-0.0000000000000000000000000001 (+/-1E-28).

Data Types

Decimal

You might need to use the D type character to assign a large value to a Decimal variable or constant.

```
Dim d1, d2, d3, d4 As Decimal
d1 = 2.375D
d2 = 1.625D
d3 = d1 + d2
d4 = 4.000D
MsgBox("d1 = " & CStr(d1) & ", d2 = " & CStr(d2) &
      ", d3 = " & CStr(d3) & ", d4 = " & CStr(d4))
```

Data Types

Double

Holds signed IEEE 64-bit (8-byte) double-precision floating-point numbers that range in value from -1.79769313486231570E+308 through -4.94065645841246544E-324 for negative values and from 4.94065645841246544E-324 through 1.79769313486231570E+308 for positive values. Double-precision numbers store an approximation of a real number.

The Double data type provides the largest and smallest possible magnitudes for a number.

The default value of Double is 0.

Appending the literal type character R to a literal forces it to the Double data type. For example, if an integer value is followed by R, the value is changed to a Double.

```
Dim dub As Double = 4.0R
```

Data Types

Double

Appending the identifier type character # to any identifier forces it to Double. In the following example, the variable num is typed as a Double:

```
Dim num# = 3
```

When you work with floating-point numbers, remember that they do not always have a precise representation in memory. This could lead to unexpected results from certain operations, such as value comparison and the Mod operator.

Data Types

Single

Holds signed IEEE 32-bit (4-byte) single-precision floating-point numbers ranging in value from -3.4028235E+38 through -1.401298E-45 for negative values and from 1.401298E-45 through 3.4028235E+38 for positive values. Single-precision numbers store an approximation of a real number.

The default value of Single is 0.

Appending the literal type character F to a literal forces it to the Single data type. Appending the identifier type character ! to any identifier forces it to Single.

```
Dim sgle As Single = 4.0F
Dime sgle! = 4.0
```

Data Types

Integer

Holds signed 32-bit (4-byte) integers that range in value from -2,147,483,648 through 2,147,483,647.

The default value of Integer is 0.

```
' The valid range of an Integer variable is -2147483648 through +2147483647.
Dim k As Integer
k = 7
' The following statement sets k to 6.
k = 5.9
' The following statement sets k to 4
k = 4.5
' The following statement sets k to 6
' Note, Visual Basic uses banker's rounding (toward nearest even number)
k = 5.5
```

Data Types

Long

Holds signed 64-bit (8-byte) integers ranging in value from -9,223,372,036,854,775,808 through 9,223,372,036,854,775,807 (9.2...E+18).

Use the Long data type to contain integer numbers that are too large to fit in the Integer data type.

The default value of Integer is 0.

Appending the literal type character L to a literal forces it to the Long data type. Appending the identifier type character & to any identifier forces it to Long.

```
Dim lng As Long = 5L
Dim lng2& = 5
```

Data Types

Short

Holds signed 16-bit (2-byte) integers that range in value from -32,768 through 32,767.

Use the Short data type to contain integer values that do not require the full data width of Integer. In some cases, the common language runtime can pack your Short variables closely together and save memory consumption.

The default value of Short is 0.

Appending the literal type character S to a literal forces it to the Short data type. Short has no identifier type character.

```
Dim sht As Short = 5S
```

Data Types

String

Holds sequences of unsigned 16-bit (2-byte) code points that range in value from 0 through 65535. Each code point, or character code, represents a single Unicode character. A string can contain from 0 to approximately two billion (2 ^ 31) Unicode characters.

Use the String data type to hold multiple characters without the array management overhead of Char(), an array of Char elements.

The default value of String is Nothing (a null reference). Note that this is not the same as the empty string (value "").

```
Dim h As String = "Hello"
```

Data Types

Object

Holds addresses that refer to objects. You can assign any reference type (string, array, class, or interface) to an Object variable. An Object variable can also refer to data of any value type (numeric, Boolean, Char, Date, structure, or enumeration).

The Object data type can point to data of any data type, including any object instance your application recognizes.

Use Object when you do not know at compile time what data type the variable might point to.

The default value of Object is Nothing (a null reference).

```
Dim objDb As Object
Dim myCollection As New Collection()
' Suppose myCollection has now been populated.
objDb = myCollection.Item(1)
```

Data Types

User-Defined

Holds data in a format you define. The Structure statement defines the format.

Define and use a structure data type when you need to combine various data types into a single unit, or when none of the elementary data types serve your needs.

The default value of a structure data type consists of the combination of the default values of each of its members.

```
[Public | Protected | Friend | Protected Friend | Private] Structure
structname
   {Dim | Public | Friend | Private} member1 As datatype1
   ' ...
   {Dim | Public | Friend | Private} memberN As datatypeN
End Structure
```

Data Types

User-Defined

```
Public Structure employee
    Public firstName As String
    Public middleName As String
    Public lastName As String
    Private salary As Double
    Private bonus As Double
    Friend Sub calculateBonus(ByVal rate As Single)
        bonus = salary * CDbl(rate)
    End Sub
End Structure
```

Type Conversion Functions

CBool(expression)
CByte(expression)
CChar(expression)
CDate(expression)
CDbl(expression)
CDec(expression)
CInt(expression)
CLng(expression)

CObj(expression)
CSByte(expression)
CShort(expression)
CSng(expression)
CStr(expression)
CUInt(expression)
CULng(expression)
CUShort(expression)

Type Conversion Functions

Example

```
Dim aDbl As Double
Dim anInt As Integer
aDbl = 2345.5678
' The following line of code sets an Int to 2346.
anInt = CInt(aDbl)
```

```
Dim aDouble As Double
Dim aString As String
aDouble = 437.324
' The following line of code sets aString to "437.324".
aString = CStr(aDouble)
```

Type Conversion Functions

To…(whatever)

- ToBoolean
- ToByte
- ToChar
- ToCharArrayRankOne
- ToDate
- ToDecimal
- ToDouble
- ToGenericParameter(T)
- ToInteger

- ToLong
- ToSByte
- ToShort
- ToSingle
- ToString
- ToUInteger
- ToULong
- ToUShort

Type Conversion Functions

To…(whatever)

```
Dim dates() As Date = { #07/14/2009#, #6:32PM#, #02/12/2009 7:16AM#}
Dim result As String


For Each dateValue As Date In dates
   result = Convert.ToString(dateValue)
   Console.WriteLine("Converted the {0} value {1} to the {2} value {3}.", _
                     dateValue.GetType().Name, dateValue, _
                     result.GetType().Name, result)
Next
```

Casting

CType

Returns the result of explicitly converting an expression to a specified data type, object, structure, class, or interface.

CType(*expression*, *typename*)

expression

- Any valid expression. If the value of expression is outside the range allowed by typename, Visual Basic throws an exception.

typename

- Any expression that is legal within an As clause in a Dim statement, that is, the name of any data type, object, structure, class, or interface.

Casting

CType

```
Dim testNumber As Long = 1000

' The following line of code sets testNewType to 1000.0.

Dim testNewType As Single = CType(testNumber, Single)
```