Chapter 15: Data Types and Limits

Section 15.1: Variant

```
Dim Value As Variant    'Explicit
Dim Value               'Implicit
```

A Variant is a COM data type that is used for storing and exchanging values of arbitrary types, and any other type in VBA can be assigned to a Variant. Variables declared without an explicit type specified by `As [Type]` default to Variant.

Variants are stored in memory as a VARIANT structure that consists of a byte type descriptor (VARTYPE) followed by 6 reserved bytes then an 8 byte data area. For numeric types (including Date and Boolean), the underlying value is stored in the Variant itself. For all other types, the data area contains a pointer to the underlying value.

VARTYPE		Reserved						Data area			
0	1	2	3	4	5	6	7	8	9	10	11

The underlying type of a Variant can be determined with either the `VarType()` function which returns the numeric value stored in the type descriptor, or the `TypeName()` function which returns the string representation:

```
Dim Example As Variant
Example = 42
Debug.Print VarType(Example)    'Prints 2 (VT_I2)
Debug.Print TypeName(Example)   'Prints "Integer"
Example = "Some text"
Debug.Print VarType(Example)    'Prints 8 (VT_BSTR)
Debug.Print TypeName(Example)   'Prints "String"
```

Because Variants can store values of any type, assignments from literals without type hints will be implicitly cast to a Variant of the appropriate type according to the table below. Literals with type hints will be cast to a Variant of the hinted type.

Value	Resulting type
String values	String
Non-floating point numbers in Integer range	Integer
Non-floating point numbers in Long range	Long
Non-floating point numbers outside of Long range	Double
All floating point numbers	Double

Note: Unless there is a specific reason to use a Variant (i.e. an iterator in a For Each loop or an API requirement), the type should generally be avoided for routine tasks for the following reasons:

- They are not type safe, increasing the possibility of runtime errors. For example, a Variant holding an Integer value will silently change itself into a Long instead of overflowing.
- They introduce processing overhead by requiring at least one additional pointer dereference.
- The memory requirement for a Variant is always ***at least*** 8 bytes higher than needed to store the underlying type.

The casting function to convert to a Variant is `CVar()`.

Section 15.2: Boolean

```
Dim Value As Boolean
```

A Boolean is used to store values that can be represented as either True or False. Internally, the data type is stored as a 16 bit value with 0 representing False and any other value representing True.

It should be noted that when a Boolean is cast to a numeric type, all of the bits are set to 1. This results in an internal representation of -1 for signed types and the maximum value for an unsigned type (Byte).

```
Dim Example As Boolean
Example = True
Debug.Print CInt(Example)   'Prints -1
Debug.Print CBool(42)       'Prints True
Debug.Print CByte(True)     'Prints 255
```

The casting function to convert to a Boolean is `CBool()`. Even though it is represented internally as a 16 bit number, casting to a Boolean from values outside of that range is safe from overflow, although it sets all 16 bits to 1:

```
Dim Example As Boolean
Example = CBool(2 ^ 17)
Debug.Print CInt(Example)   'Prints -1
Debug.Print CByte(Example)  'Prints 255
```

Section 15.3: String

A String represents a sequence of characters, and comes in two flavors:

Variable length

```
Dim Value As String
```

A variable length String allows appending and truncation and is stored in memory as a COM BSTR. This consists of a 4 byte unsigned integer that stores the length of the String in bytes followed by the string data itself as wide characters (2 bytes per character) and terminated with 2 null bytes. Thus, the maximum string length that can be handled by VBA is 2,147,483,647 characters.

The internal pointer to the structure (retrievable by the `StrPtr()` function) points to the memory location of the *data*, not the length prefix. This means that a VBA String can be passed directly API functions that require a pointer to a character array.

Because the length can change, VBA reallocates memory for a String *every time the variable is assigned to*, which can impose performance penalties for procedures that alter them repeatedly.

Fixed length

```
Dim Value As String * 1024    'Declares a fixed length string of 1024 characters.
```

Fixed length strings are allocated 2 bytes for each character and are stored in memory as a simple byte array. Once allocated, the length of the String is immutable. They are ***not*** null terminated in memory, so a string that fills the memory allocated with non-null characters is unsuitable for passing to API functions expecting a null terminated string.

Fixed length strings carry over a legacy 16 bit index limitation, so can only be up to 65,535 characters in length. Attempting to assign a value longer than the available memory space will not result in a runtime error - instead the resulting value will simply be truncated:

```
Dim Foobar As String * 5
Foobar = "Foo" & "bar"
Debug.Print Foobar          'Prints "Fooba"
```

The casting function to convert to a String of either type is `CStr()`.

Section 15.4: Byte

```
Dim Value As Byte
```

A Byte is an unsigned 8 bit data type. It can represent integer numbers between 0 and 255 and attempting to store a value outside of that range will result in runtime error 6: `Overflow`. Byte is the only intrinsic unsigned type available in VBA.

The casting function to convert to a Byte is `CByte()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up.

Byte Arrays and Strings

Strings and byte arrays can be substituted for one another through simple assignment (no conversion functions necessary).

For example:

```
Sub ByteToStringAndBack()

Dim str As String
str = "Hello, World!"

Dim byt() As Byte
byt = str

Debug.Print byt(0)  ' 72

Dim str2 As String
str2 = byt

Debug.Print str2    ' Hello, World!

End Sub
```

In order to be able to encode Unicode characters, each character in the string takes up two bytes in the array, with the least significant byte first. For example:

```
Sub UnicodeExample()

Dim str As String
str = ChrW(&H2123) & "."  ' Versicle character and a dot

Dim byt() As Byte
byt = str

Debug.Print byt(0), byt(1), byt(2), byt(3)  ' Prints: 35,33,46,0

End Sub
```

Section 15.5: Currency

```
Dim Value As Currency
```

A Currency is a signed 64 bit floating point data type similar to a Double, but scaled by 10,000 to give greater precision to the 4 digits to the right of the decimal point. A Currency variable can store values from -922,337,203,685,477.5808 to 922,337,203,685,477.5807, giving it the largest capacity of any intrinsic type in a 32 bit application. As the name of the data type implies, it is considered best practice to use this data type when representing monetary calculations as the scaling helps to avoid rounding errors.

The casting function to convert to a Currency is `CCur()`.

Section 15.6: Decimal

```
Dim Value As Variant
Value = CDec(1.234)

'Set Value to the smallest possible Decimal value
Value = CDec("0.0000000000000000000000000001")
```

The `Decimal` data-type is *only* available as a sub-type of `Variant`, so you must declare any variable that needs to contain a `Decimal` as a `Variant` and *then* assign a `Decimal` value using the `CDec` function. The keyword `Decimal` is a reserved word (which suggests that VBA was eventually going to add first-class support for the type), so `Decimal` cannot be used as a variable or procedure name.

The `Decimal` type requires 14 bytes of memory (in addition to the bytes required by the parent Variant) and can store numbers with up to 28 decimal places. For numbers without any decimal places, the range of allowed values is -79,228,162,514,264,337,593,543,950,335 to +79,228,162,514,264,337,593,543,950,335 inclusive. For numbers with the maximum 28 decimal places, the range of allowed values is -7.9228162514264337593543950335 to +7.9228162514264337593543950335 inclusive.

Section 15.7: Integer

```
Dim Value As Integer
```

An Integer is a signed 16 bit data type. It can store integer numbers in the range of -32,768 to 32,767 and attempting to store a value outside of that range will result in runtime error 6: Overflow.

Integers are stored in memory as little-endian values with negatives represented as a two's complement.

Note that in general, it is better practice to use a Long rather than an Integer unless the smaller type is a member of a Type or is required (either by an API calling convention or some other reason) to be 2 bytes. In most cases VBA treats Integers as 32 bit internally, so there is usually no advantage to using the smaller type. Additionally, there is a performance penalty incurred every time an Integer type is used as it is silently cast as a Long.

The casting function to convert to an Integer is `CInt()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up.

Section 15.8: Long

```
Dim Value As Long
```

A Long is a signed 32 bit data type. It can store integer numbers in the range of -2,147,483,648 to 2,147,483,647 and

attempting to store a value outside of that range will result in runtime error 6: Overflow.

Longs are stored in memory as little-endian values with negatives represented as a two's complement.

Note that since a Long matches the width of a pointer in a 32 bit operating system, Longs are commonly used for storing and passing pointers to and from API functions.

The casting function to convert to a Long is `CLng()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up.

Section 15.9: Single

```
Dim Value As Single
```

A Single is a signed 32 bit floating point data type. It is stored internally using a little-endian IEEE 754 memory layout. As such, there is not a fixed range of values that can be represented by the data type - what is limited is the precision of value stored. A Single can store a value ***integer*** values in the range of -16,777,216 to 16,777,216 without a loss of precision. The precision of floating point numbers depends on the exponent.

A Single will overflow if assigned a value greater than roughly 2128. It will not overflow with negative exponents, although the usable precision will be questionable before the upper limit is reached.

As with all floating point numbers, care should be taken when making equality comparisons. Best practice is to include a delta value appropriate to the required precision.

The casting function to convert to a Single is `CSng()`.

Section 15.10: Double

```
Dim Value As Double
```

A Double is a signed 64 bit floating point data type. Like the Single, it is stored internally using a little-endian IEEE 754 memory layout and the same precautions regarding precision should be taken. A Double can store ***integer*** values in the range of -9,007,199,254,740,992 to 9,007,199,254,740,992 without a loss of precision. The precision of floating point numbers depends on the exponent.

A Double will overflow if assigned a value greater than roughly 21024. It will not overflow with negative exponents, although the usable precision will be questionable before the upper limit is reached.

The casting function to convert to a Double is `CDbl()`.

Section 15.11: Date

```
Dim Value As Date
```

A Date type is represented internally as a signed 64 bit floating point data type with the value to the left of the decimal representing the number of days from the epoch date of December 30th, 1899 (although see the note below). The value to the right of the decimal represents the time as a fractional day. Thus, an integer Date would have a time component of 12:00:00AM and x.5 would have a time component of 12:00:00PM.

Valid values for Dates are between January 1st 100 and December 31st 9999. Since a Double has a larger range, it is possible to overflow a Date by assigning values outside of that range.

As such, it can be used interchangeably with a Double for Date calculations:

```
Dim MyDate As Double
MyDate = 0                                  'Epoch date.
Debug.Print Format$(MyDate, "yyyy-mm-dd")   'Prints 1899-12-30.
MyDate = MyDate + 365
Debug.Print Format$(MyDate, "yyyy-mm-dd")   'Prints 1900-12-30.
```

The casting function to convert to a Date is `CDate()`, which accepts any numeric type string date/time representation. It is important to note that string representations of dates will be converted based on the current locale setting in use, so direct casts should be avoided if the code is meant to be portable.

Section 15.12: LongLong

```
Dim Value As LongLong
```

A LongLong is a signed 64 bit data type and is only available in 64 bit applications. It is **not** available in 32 bit applications running on 64 bit operating systems. It can store integer values in the range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 and attempting to store a value outside of that range will result in runtime error 6: Overflow.

LongLongs are stored in memory as little-endian values with negatives represented as a two's complement.

The LongLong data type was introduced as part of VBA's 64 bit operating system support. In 64 bit applications, this value can be used to store and pass pointers to 64 bit APIs.

The casting function to convert to a LongLong is `CLngLng()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up.

Section 15.13: LongPtr

```
Dim Value As LongPtr
```

The LongPtr was introduced into VBA in order to support 64 bit platforms. On a 32 bit system, it is treated as a Long and on 64 bit systems it is treated as a LongLong.

It's primary use is in providing a portable way to store and pass pointers on both architectures (See Changing code behavior at compile time.

Although it is treated by the operating system as a memory address when used in API calls, it should be noted that VBA treats it like signed type (and therefore subject to unsigned to signed overflow). For this reason, any pointer arithmetic performed using LongPtrs should not use > or < comparisons. This "quirk" also makes it possible that adding simple offsets pointing to valid addresses in memory can cause overflow errors, so caution should be taken when working with pointers in VBA.

The casting function to convert to a LongPtr is `CLngPtr()`. For casts from floating point types, the result is rounded to the nearest integer value with .5 rounding up (although since it is usually a memory address, using it as an assignment target for a floating point calculation is dangerous at best).