String Substitute
Replace(strVariable, <substring1>, <substring2>)
e.g. standardize street address
1234 S.W. 23 Street -> 1234 SW 23rd St
strAddr = "1234 S.W. 23 Street"
Replace(strAddr, "S.W.", "SW")
Replace(strAddr, "N.W.", "NW")
Replace(strAddr, "23", "23rd")
Replace(strAddr, "Street", "St")

Manipulating strings - Mid$
Mid$(string, start[, length])
MyString = "Mid Function Demo" ' Create text string (17 chars)
FirstWord = Mid(MyString, 1, 3) ' Returns "Mid".
LastWord = Mid(MyString, 14, 4) ' Returns "Demo".
MidWords = Mid(MyString, 5) ' Returns "Function Demo"
allWords = Mid(MyString, 4, 20) ' Returns " Function Demo"
allWords = Mid(MyString, 25) ' Returns ""

Reverse a string
• Returns a string in which the character order of a specified string is reversed
StrReverse(string)
StrReverse("abcdefg") -> "gfedcba"

Converting string to upper and lower cases
• Returns a Variant (String) containing the specified string, converted to uppercase
UCase(string)
e.g.
Dim UpperCase, LowerCase
UpperCase = "Hello World 1234" ' String to convert.
LowerCase = Lower(UpperCase) ' Returns "hello world 1234".

Conversion between ASCII code and character
Arc("a") ' Returns 97
Char$(97) ' Returns “a"
IsNumeric - a number?

Dim MyVar, MyCheck ' variants
MyVar = "53" ' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns True.
MyVar = "459.95" ' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns True.
MyVar = "45 Help" ' Assign value.
MyCheck = IsNumeric(MyVar) ' Returns False.

Val - converts strings to numbers

• Blanks, tabs, and linefeed characters are stripped
• $ and comma not recognized

Dim MyValue
MyValue = Val("2457") ' Returns 2457.
MyValue = Val(" 2 45 7") ' Returns 2457.
MyValue = Val("24 and 57") ' Returns 24.
MyValue = Val(" 1615 198th Street N.E.") ' Returns 1615198
MyValue = Val("&HFFFF") ' Returns hexadecimal value -1
MyValue = Val("ABCD") ' Returns 0

Str$ - convert numbers to strings

Str(number) ' Returns “1234”
• a leading space is always reserved for the sign of number.
Plus sign is implied by a leading space.
• recognizes only the period (.) as a valid decimal separator.
Use CStr when different decimal separators may be used
(for example, in international applications).
• use Format function to convert numeric values
representing dates, times, or currency

Type conversion functions

• Read Section 8.15.4
• Operate on string or numeric expressions
  CBool  convert to boolean
  CByte  convert to byte (number between 0 and 255)
  CChar  convert to currency
  CDate  convert to date
  CDbl  convert to double
  CDec  convert to decimal
  CInt  convert to integer
  CLng  convert to long
  CSng  convert to single
  CStr  convert to string
  CVar  convert to variant

String Formatting

• Returns a Variant (String) containing a formatted expression according to a format expression.
  Format(expression, format, firstdayofweek[, firstweekofyear][])
expression (required) - any valid expression.
format (optional) - a valid named or user-defined format expression.
firstdayofweek (optional) - a constant specifying the first day of the week.
Firstweekofyear (optional) - a constant that specifies the first week of the year.

Named generic formats

• General number - no thousand separators
• Currency - $12,035.25 (depends on locale setting)
• Fixed - 1 digit left, 2 right of the decimal point
• Standard - fixed with thousand separator(s)
• Percent - * 100 and append % to right
• Scientific - standard scientific notation
• Yes/No - No if number is 0; otherwise Yes
• True/False - False if number is 0; otherwise True
• On/Off - Off if number is 0; otherwise On
Named generic formats - examples

- Format(1, “True/False”) -> True
- Format(0, “On/Off”) -> Off
- Format(1, “Yes/No”) -> Yes
- Format(0.05475, “Percent”) -> 5.475%

User defined formats

- 0 - displays a digit at the position of 0
- # - digit place holder, displays no digits if n/a
- , - thousand separator
- $ - shows dollar sign

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Format(0.123, “0.00%”)</td>
<td>12.30%</td>
</tr>
<tr>
<td>Format(0.123, “0%”)</td>
<td>12%</td>
</tr>
<tr>
<td>Format(-1123456.789, “$#,##0.00”)</td>
<td>-1,123,456.79</td>
</tr>
<tr>
<td>Format(-1123456.789, “0.00E+00”)</td>
<td>-1.12E+06</td>
</tr>
<tr>
<td>Format(-1123456.789, “0.00E-00”)</td>
<td>-1.12E06</td>
</tr>
</tbody>
</table>

FormatNumber function

FormatNumber(Expression, DigitsAfterDecimal
[IncludeLeadingDigit, UseParensForNegativeNumbers, GroupDigits]])
- Expression (required) - expression to be formatted.
- DigitsAfterDecimal (optional) - number of places to the right of the decimal. Default value –1 indicates use of computer's regional settings.
- IncludeLeadingDigit (optional) - Tristate constant indicating if a leading zero is displayed for fractional values.
- UseParensForNegativeNumbers (optional) - Tristate constant indicates if to place negative values within parentheses.
- GroupDigits (optional) - Tristate constant indicates if numbers are grouped using the group delimiter specified in the computer's regional settings.

Tristate constant

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TristateTrue</td>
<td>–1</td>
<td>True</td>
</tr>
<tr>
<td>TristateFalse</td>
<td>0</td>
<td>False</td>
</tr>
<tr>
<td>TristateUseDefault</td>
<td>–2</td>
<td>Use the setting from the computer's regional settings.</td>
</tr>
</tbody>
</table>

FormatCurrency function

FormatCurrency(Expression, NumDigitsAfterDecimal
[IncludeLeadingDigit, UseParensForNegativeNumbers, GroupDigits]])
- NumDigitsAfterDecimal (optional) - number of places to the right of the decimal displayed. Default value –1 (use computer's regional settings).
- IncludeLeadingDigit (optional) - Tristate constant indicating if a leading zero is displayed for fractional values.
- UseParensForNegativeNumbers (optional) - Tristate constant indicating if to place negative values within parentheses.
- GroupDigits (optional) - Tristate constant indicating if grouping number using group delimiter specified in computer's regional settings.

FormatPercent function

FormatPercent(Expression, NumDigitsAfterDecimal
[IncludeLeadingDigit, UseParensForNegativeNumbers, GroupDigits]])
- NumDigitsAfterDecimal (optional) - number of places to the right of the decimal. Default value –1 (use computer's regional settings).
- IncludeLeadingDigit (optional) - Tristate constant indicating if a leading zero is displayed for fractional values.
- UseParensForNegativeNumbers (optional) - Tristate constant indicating if to place negative values within parentheses.
- GroupDigits (optional) - Tristate constant indicating if grouping numbers using delimiter specified in computer's regional settings.
Date and time processing

- Now - current date and time
- Date - today's date
- Day - consecutive day in the year
- Weekday - weekday in number
- WeekdayName - name of weekday
- Month - month as number
- MonthName(date, True) - month name abbreviated
- MonthName(date) - month name unabbreviated
- Year - year
- #3/2/1987# - date literal, implicitly converted to date format of the locale specified in Control Panel's Regional Setting

Date and time processing

- IsDate(expression) - returns a boolean
  expression (required) - variant containing a date expression or a string expression recognizable as a date or time.
- DateValue(date) returns a variant (date)
  DateValue(2-15-73) -> 2/15/73
  DateValue(February 15, 1973) -> 2/15/1973
- DateSerial(year, month, day) - Returns a Variant (Date) for a specified year, month, and day. Used to create a date.
  DateSerial(1998, 8, 2) -> 8/2/98

Date and time processing

DatePart(interval, date, [firstdayofweek], [firstweekofyear]))

- interval (required) - string expression, the interval of time you want to return.
- date (required) - Variant (Date), value to evaluate.
- Firstdayofweek (optional) - constant specifying first day of week. (Default is Sunday).
- Firstweekofyear (optional) - constant specifying first week of year. (Default the week in which January 1 occurs).

Settings - Interval

| yyyy | Year | 'DatePart("yyyy", Now) -> 2000 |
| q    | Quarter | 'DatePart("q", Now) -> 3 |
| m    | Month | 'DatePart("m", Now) -> 10 |
| y    | Day of year | 'DatePart("y", Now) -> 290 |
| d    | Day | 'DatePart("d", Now) -> 16 |
| w    | Weekday | 'DatePart("w", Now) -> 2 |
| ww   | Week | 'DatePart("ww", Now) -> 43 |
| h    | Hour | 'DatePart("h", Now) -> 17 |
| n    | Minute | 'DatePart("n", Now) -> 50 |
| s    | Second | 'DatePart("s", Now) -> 15 |

Settings - firstdayofweek

| vbUseSystem | Use the NLS API setting. |
| vbSunday    | 1 Sunday (default) |
| vbMonday    | 2 Monday |
| vbTuesday   | 3 Tuesday |
| vbWednesday | 4 Wednesday |
| vbThursday  | 5 Thursday |
| vbFriday    | 6 Friday |
| vbSaturday  | 7 Saturday |

Settings - firstweekofyear

| vbUseSystem | Use the NLS API setting. |
| vbFirstJan1 | 1 Start with week in which January 1 occurs (default). |
| vbFirstFourDays | 2 Start with the first week that has at least four days in the new year. |
| vbFirstFullWeek | 3 Start with first full week of the year. |
Get past or future dates - DateAdd

DateAdd(interval, number, date)

- **interval** (required) - string expression, the interval of time to add
- **number** (required) - numeric expression, the number of intervals you want to add. It can be positive (to get dates in the future) or negative (to get dates in the past).
- **date** (required) - Variant (Date) or literal representing date to which the interval is added.

DateAdd("yyyy", 3, Now) -> 10/16/03 12:31:59 PM
DateAdd("m", 5, Now) -> 3/16/01 12:31:59 PM
DateAdd("d", -17, Now) -> 9/29/00 12:31:59 PM

Time Elapse - DateDiff

DateDiff(interval, date1, date2)

- **interval** (required) - the interval of time use to calculate the difference between **date1** and **date2**.
- **date1**, **date2** (required) - Variant (Date). Two dates to use in the calculation.
- **Firstdayofweek** (optional) - the first day of the week. If not specified, Sunday is assumed.
- **Firstweekofyear** (optional) - the first week of the year. If not specified, the first week is assumed to be the week in which January 1 occurs.

Time processing

- Time
- Hour
- Minute
- Second
- Time
- TimeValue
- TimeSerial

Date and time formatting

- **FormatDateTime** (Date[,NamedFormat])

  - Named format
    - vbGeneralDate
    - vbLongDate
    - vbShortDate
    - vbLongTime
    - vbShortTime

FormatDateTime(Now, "General Date") -> 8/3/98 5:14:12 PM
FormatDateTime(Now, "Long Date") -> Monday, August 03, 1998
FormatDateTime(Now, "Short Date") -> 8/3/98
FormatDateTime(Now, "Long Time") -> 5:14:00 PM
FormatDateTime(Now, "Short Time") -> 17:14
FormatDateTime(Now, "Medium Time") -> 05:14 PM
FormatDateTime(Now, "Hh:nn:SS AM/PM") -> 05:16:50 PM
FormatDateTime(Now, "tttt") -> 5:16:50 PM
Date application example

- A program uses a file to perform certain tasks and produce results.
- Because processing the file is time consuming, unless the file contents have changed, there is no need to rerun the code associated with file processing.
- How does the program know if an update is needed?

Date application example

- use a text file to record date when file processing was performed last time
- run program and check if the file date is the same as that in the text file. If so, no need to process file. Otherwise, file needs to be read and processed.
- Check the date file is last created or modified
  $\text{FileDateTime(pathname)}$  \quad \text{getting the date stamp}$