

CHAPTER 8 PART II

STRING\$ AND SPACES

STRING\$ AND SPACES

Create strings of specified number

String\$ creates string of specified character

Space\$ creates string of spaces

Program: Figure 8.8, p. 317

Example: string\$(10, "A") -creates 10 A's

string\$(5, 97) -creates 5 a's

space\$(5) & string\$(5, "a") -creates 5 spaces which are appended to 5 a's

FUNCTION REPLACE

Search string and replace with another string

Example: lblOutput.Caption =
Replace(txtInput.Text, " ", ".")

replace blanks in txtInput.Text with 2 periods

optional arguments:

- (4) indicate starting character position for search
- (5) specify number of replacements to perform
- (6) specifies comparison type -binary or text

Figure 8.2, p. 309

FUNCTIONS STRREVERSE, UCASE, AND LCASE

Reverses a string

Example: txtOutput.Text =
StrReverse(txtInput.Text)

UCase converts string to upper case

LCase converts string to lower case

Examples:

txtUpperCase.Text = UCase(txtInput.Text)

txtLowerCase.Text = LCase(txtInput.Text)

Figure 8.10 and Figure 8.11

CONVERSION FUNCTIONS: ASC AND CHR\$

Asc returns ASCII code corresponding to a given character

Example: Asc("a") returns 97

Note: if contains more than one character, returns ASCII code of first character only

Chr\$ returns string corresponding to ASCII code

Example: Chr\$(34) returns " (double quotes)

Figure 8.12 demonstrates these functions

CONVERSION FUNCTIONS: ISNUMERIC, VAL, AND STR\$

IsNumeric- returns True if string is numeric

Val-converts strings to numbers

Str\$-converts numbers to strings

Example: If IsNumeric(txtInput.Text) Then

lblOutput.Caption = txtInput.Text & "
+ 10 is " & Str\$(Val(txtInput.Text) + 10)

Note1: valid input to Val: 0-9, + or -, and .

Note2: convert hexadecimal value preceded by &H

Note3: convert octal values preceded by &O

FUNCTIONS: HEX\$ AND OCT\$

Hex\$- converts numbers to hexadecimal (base 16) format strings

Oct\$- converts numbers to octal (base 8) format strings

Example:

```
txtHex.Text = Hex$(txtInput.Text)
```

```
txtOct.Text = Oct$(txtInput.Text)
```

Code example: Figure 8.14, p. 323-324

TYPE CONVERSION FUNCTIONS

Note: conversion errors occur at run-time

Cbool- convert string to Boolean

(1) if numeric zero or string zero, returns False

(2) all other numeric values or strings returns

True

Cbyte- convert string to value 0 to 255

nonnumeric values or strings result in error

Ccur- convert string to Currency

nonnumeric values or strings result in error

TYPE CONVERSION FUNCTIONS

Cdate-convert string to a date value
(numeric or string representation)

CDbl-convert string to a Double value
(double-precision, floating point)

CDec-convert string to Decimal value

CInt-convert string to Integer value

CLng-convert string to Long Integer value

Cvar-convert string to Variant : numeric=Double,
non-numeric=string

STRING FORMATTING

CStr-convert string depending on type:

(1)date=short format date returned

(2)boolean=True or False returned

(3)number=corresponding number returned

(4)error=corresponding error message returned

(5)null=error occurs

String Formatting:

Format\$ "General Number"-displays number with
no separators for places beyond hundreds

STRING FORMATTING

Format\$ "Currency"-number with dollar symbol,
separator for thousands, and two digits after
decimal

Format\$ "Fixed"-number with at least one digit to
left of decimal, and two to right

Format\$ "Standard"-separators for thousands, at
least one digit to left of decimal and two to right

Format\$ "Scientific"-scientific notation with two
digits to right of decimal

STRING FORMATTING

Format\$ "Percent"-multiplies number by 100,
displays % to right and two digits right of decimal

Format\$ "Yes/No"-displays No if 0, Yes otherwise

Format\$ "True/False"-False if 0, Yes otherwise

Format\$ "On/Off"-Off if 0, On otherwise

Coded example: Figure 8.16, p. 328

Formats are formed using strings containing
formatting flags: Figure 8.17, p. 329-330

Lines 6-10, no formatting of numbers

STRING FORMATTING

Lines 11-13 use format flag 0 which represents a required digit and displayed as whole number (floating-point values are automatically rounded)
Lines 14-18 use format flag 0 with "0.00" to insure at least one digit to left and two to right of decimal
Lines 19-24 use format flag # with "#,##0.00"
represents a digit place holder, if no digit present nothing is displayed; flag , invoked for thousands; two decimal places to right

STRING FORMATTING

Lines 25-30 uses "\$,##0.00" which is same format except uses \$ literal; valid literals: -, +, (,)
Note: a backslash (\) will allow other literals to be displayed that follow ; "\@#,##0.00"
Lines 31-33 uses "0%" output whole number percent
Lines 34-39 uses "0.00%" output percent with two decimals to right
Lines 40-45 uses "0.00E+00" scientific notation with at least two digits in the exponent; negative exponents displayed with - sign

STRING FORMATTING

Lines 46-51 demonstrate negative exponents
Lines 52-55 demonstrate formats for non-negative and negative values: "\$,##0.00;(\$,##0.00)"
Note: semi-colons used to separate formats and up to four formats can be used where third format is for zero values and the fourth is for null values
Function FormatNumber- used to format values
Code in Figure 8.18, p. 333
Arguments: (1) numeric expression, (2) digits right of decimal, (3) display leading zero or not, (4) whether or not parentheses displayed around negative numbers, (5) display or not thousands place separators (2-5 are optional)

STRING FORMATTING

Function FormatCurrency:
Code in Figure 8.19, p. 336
Arguments:
(1) numeric value to format
(2) number of digits to right of decimal
(3) whether or not leading zero displayed
(4) whether or not parentheses are placed around negative numbers
(5) whether or not thousands separators are used (2-5 are optional)

STRING FORMATTING

Function FormatPercent:
Code in Figure 8.20, p. 338
Arguments:
(1) numeric value to format
(2) number of digits to right of decimal
(3) whether or not leading zero displayed
(4) whether or not parentheses are placed around negative numbers
(5) whether or not thousands separators are used (2-5 are optional)

DATE AND TIME PROCESSING

Function Now- current system date and time
Example: "Current data and time: " & Now
Function Date- current system date
Example: "Date: " & Date
Function Day- gives day of month 1-31
Example: "Day: " & Day(Date)
Function WeekDay- day of week as integer (1-7)
Example: "Weekday: " & WeekDay(Date)
by default Sunday=1

DATE AND TIME PROCESSING

Example2: "WeekdayName: " & WeekdayName(Weekday(Date)) where WeekdayName returns the string name of weekday

Example3: "WeekdayName: " & WeekdayName(Weekday(Date), True)

True indicates the abbreviated name should be returned (i.e. Sunday, Sun)

Example4: "Month: " & Month(Date) returns 1-12

Example5: "MonthName: " & MonthName(Month(Date)) returns string name with January=1

DATE AND TIME PROCESSING

Example6: "MonthName abbreviated: " & MonthName(Month(Date), True) where True returns the short name (i.e. Jan=January)

Example7: "Year: " & Year(Date) returns year as integer

Example8: #3/2/1996# is a data literal and enclosed within two pound signs

IsDate-use to determine if string can be converted to date (code in Figure 8.22, p. 342-343)

Example: Format (IsDate(123456), "True/False")

DATE AND TIME PROCESSING

Note: the function recognizes dates in the range January 1, 100 to December 31, 9999

Viable formats:

January 1, 1999

Jan 1, 1999

1/1/1999

1/1/99

1-Jan-99

1-Jan-1999

DATE AND TIME PROCESSING

Function DateValue-convert a string into a date

Example: "DateValue("2-15-73")

Viable formats:

January 1, 1999

Jan 1, 1999

1/1/1999

1/1/99

1-Jan-99

1-Jan-1999

DATE AND TIME PROCESSING

Function DataSerial- create dates, receives three arguments: year, month, and day

Code example: Figure 8.22, p. 342-343

Example: DateSerial(1998, 8, 2)

Function DatePart-receives two arguments: (1) a string to indicate part of date to return, (2) data expression

Example: DatePart("yyyy", Now)-get year from current date and time returned by Now

Code example: Figure 8.23, p. 344

DATE AND TIME PROCESSING

Figure 8.24, p. 346 demonstrates adding and subtracting dates

Function DateAdd- add to a date and has three arguments: (1) interval, (2) value to add, & (3) date interval is string indicating part of date modified string indicates 2nd argument should be added to year

Function DateDiff- subtract dates and has three arguments: (1) interval, (2) 1st date, & (3) 2nd date

DATE AND TIME PROCESSING

- Optional arguments: (4) constant indicating day of week, (5) constant indicating 1st week of year
- Example: DateDiff("d", "1/1/98", Now)
- Figure 8.25, p. 347 displays the following functions: Hour, Minute, Second, Timer (number of seconds since midnight)
- TimeSerial-return date containing specified time
- Arguments: (1) hour as value 0-23, (2) minute & (3) second
- example: 90= 1 hour and 30 minutes

DATE AND TIME PROCESSING

- Function TimeValue: returns date containing specified time. Range=0:00:00 AM-23:59:59 PM
- Code example: Figure 8.26, p. 348-349
- Function FormatDateTime: format current date and time as general date displaying date and time
- Arguments: (1) expression to format, (2) constant representing format
- Function Format: format current date and time
- Arguments: (1) expression to format, (2) string representing format

STRING ARRAYS

- Array whose elements are strings
particular element or string within array if referred to by giving name of string followed by index
- Example: Dim a(1) As String, b As String
- ```
a(0) = "Hello"
a(1) = "There!"
b = Join(a)
```
- Join is the concatenation function  
result="HelloThere!"

## STRING ARRAYS

- Function Filter-used in Figure 8.28 to search String array **a** for "Visual Basic 6"
- Filter returns a string array that is assigned to array **b**
- Each element of **b** stores the string "Visual Basic 6"
- Sub PrintString then prints each element of **b**
- Erase is used to erase **b**'s memory
- 3rd argument is an optional Boolean value indicates whether or not 2nd argument included in filtering
- When False, 2nd argument is excluded

## STRING ARRAYS

- Filter has a 4th optional argument determines type of comparison to use
- default is vbBinaryCompare, other values from Figure 8.2 can be used
- Visual Basic tokenization function Split is used to extract the words from a sentence
- Split extracts all substrings (or tokens)
- substring in this case is any character or group of characters separated by a space character(delimiter)
- Split(string, delimiter=" ", counter=-1, compare= vbBinaryCompare )