Chapter 3 – A Word About Printing

Back in Chapter 1, we looked at some InternetBasic programs that displayed information on the screen in the IB window. In this chapter, we'll take a look at programs that print information to printers and virtual printers. We'll cover text output now, and save graphics output for a later chapter.

The IB runtime system refers to printers by 3-character names. You can configure as many printer names as you want, and there are only two rules:

- A printer name must begin with the letter "L"
- The other two characters are alphanumeric

Here are some sample printer names:

L01
L02
L03
L04
L10
L99

A popular approach is to use "LP" as the first part of the name, followed by a number or letter, such as:

LP1
LP2
LP3
LPH
LPT

Your printer names depend on the way you configure the IB runtime system (more on this topic later).

The compact disc that accompanies this book is configured with the following printer names:

LP1 – the Windows printer that is attached to your system
LPH – a "virtual" printer that displays IB output in your browser (in HTML format)
LPT – a "virtual" printer that displays IB output in your default text editor

Your IB program must open a printer before it can send output to it. Here's how:

```
OPEN (logical-unit-number) printer-name
```

The logical-unit-number is a number that identifies the printer throughout the program. The printer-name parameter is the 3-character name of the printer to be opened (e.g., LP1, LPH, LPT). Both of these parameters can be specified with variable names (a numeric variable for the logical-unit-number, and a string variable for the printer-name).

For example, here's how to open LP1 using logical-unit-number 1:

```
OPEN (1) "LP1"
```

Here's how to use a variable for the printer-name:

```
LENGTH 3 & LOCAL PTR$                ! Define a variable

Print "Enter printer name:",@(0,3)   ! Display prompt
Input @(35,3),PTR$                   ! Input printer name

OPEN (1) PTR$                        ! Open printer
```

A logical-unit-number stays open until it is closed. This is done with the CLOSE statement:

```
CLOSE (logical-unit-number)
```

In the above example, here's how to close the printer:

```
CLOSE (1)
```

A note about logical-unit-numbers

InternetBasic supports 50 logical-unit-numbers (luns), numbered from 0 to 49. Lun 0 serves a special purpose; it is the lun for the video screen of the IB client window. Luns 1 through 49 may be used for printers, data files, and other devices that can be opened by the IB program (such as gateway devices). There are no reserved luns above 0; values 1 through 49 are chosen by the programmer.

IB provides a simply way to check for errors that might occur when a program is running. All input/output statements, such as OPEN, may contain an additional clause that redirects the program to a specified statement label in the case of a runtime error (hereinafter called an **exception**). Here's the syntax for the OPEN statement:

```
OPEN (logical-unit-number) printer-name,EXCP=statement-label
```

Here's an example:

```
OPEN (1) "LP1",EXCP=ErrorRoutine
```

In English, this is:

Open the printer named LP1, using logical-unit-number 1. If a runtime exception occurs, branch to the ErrorRoutine statement label.

What could cause a runtime exception when a program tries to open a printer? There are a couple of common reasons. Most common is that the specified printer name is not configured (e.g., you try to open LP9, but that name does not exist in your IB configuration). Also, a printer is an exclusive device that can be opened by one program at a time. If your program tries to open a printer that's already open (by another program or another IB user), your program will encounter a runtime exception.

There are specific error codes that your program can check in order to trap exceptions. We'll cover these in Chapter 6. For now, the sample programs handle all runtime exceptions in a generic manner.

Once you've opened a printer, your program can print to it by using the lun in a PRINT statement. Here's the syntax:

PRINT (lun) print-item

For example:

PRINT (1) "Customer History Report",@(0)

In English, this is:

Print the phrase "Customer History Report" on lun 1 (a printer that is open) starting at position 0 (i.e., the left-hand side of the page).

The PRINT statement may contain multiple print items, as long as a semicolon separates them. The syntax is:

PRINT (lun) print-item;print-item;print-item

For example:

PRINT (1) CUSTNUM$,@(0);NAME$,@(10);PHONE$,@(50)

In English, this is:

Print the value of the CUSTNUM$ variable starting a position 0, the value of NAME$ starting at position 10, and the value of PHONE$ starting at position 50.

Here's a complete program that shows how to open, print to, and close a printer. This program is based on one of the samples from Chapter 1.

```
! //MTB// Src(SAMPLE7.MTB,BAS) Obj(SAMPLE7,BAS)
```

```
! Define the variables

      LENGTH 2.0
      LOCAL I

      LENGTH 16.2
      LOCAL AMOUNT

      LENGTH 3
      LOCAL PRINTER$

! Here's the code

      Print (CS)                    ! Clear the screen

      ! Display program information

      Print "If you start with 1 cent and double it every day,",@(0,0)
      Print "how much money will you have in 50 days?",@(0,1)

      ! Display a prompt, then input the response

      Print "Enter printer name for the output:",@(0,3)
      Input @(35,3),PRINTER$

      Open (1) PRINTER$,EXCP=OpenError    ! Open the printer
                                          !
                                          ! If there is an exception,
                                          ! branch to the OpenError
                                          ! statement label (below).

      Print (1) "Day",@(0);"Amount",@(21) ! Print the heading

      AMOUNT = .01                        ! Set the initial amount

      For I = 1 TO 50                     ! Loop from 1 to 50
        Print (1) I,@(0);AMOUNT,@(10)     ! Print the detail line
        AMOUNT = AMOUNT * 2               ! Double the amount
      Next I                              ! Loop

      Close (1)                           ! Close the printer

      ! Display ending message and use a "dummy INPUT" to hold it there

      Print "The printout is done.",@(0,5)
      GOTO Done

OpenError:
      Print "There was an error opening the printer.",@(0,5)

Done: Print "Press ENTER to continue...",@(0,7)
      Input @(30,7),""
      STOP

END
```

As you can see, this program is named SAMPLE7 (source name is SAMPLE7.MTB). Run this program now and try printing to the virtual printer named LPH. The output is displayed in your web browser. (If you insist on using a piece of paper for the output, run the program again and choose LP1. Also, as long as you're running this program, try printing to LPT, the virtual text printer.)

The next step is to make those large numbers look better. We're going to add an **edit mask** to the output, so the large numbers will have commas between each group of 3 digits.

Since the AMOUNT variable is defined with a length and precision of 16.2, we'll have to make an edit mask that matches it. Here's what this looks like:

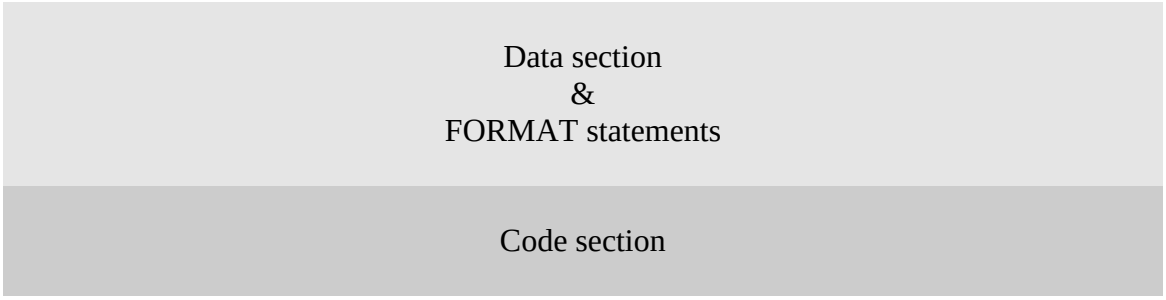
```
##,###,###,###,##0.00-
```

The digit positions are represented by # and 0, the commas represent group dividers, the decimal point serves its usual meaning, and the minus sign is the trailing character (not that this program is going to print any negative numbers, but a trailing character is required for all edit masks).

The # symbol in an edit mask tells IB to print blanks in place of leading zeroes. The 0 symbol tells IB to print the digit (zero or otherwise) starting in the position to the right of the first 0 in the edit mask. Thus, our sample edit mask always displays numbers to the right of the decimal point, but will print leading blanks up to and including the 1's place.

See Chapter ??? for more information about edit masks.

When you use an edit mask, you must use a FORMAT statement. FORMAT statements must appear before the code section of the program, and (if they include variables) after the variables have been defined.



Here's the syntax for a FORMAT statement:

```
format-label FORMAT format-item(s)
```

The format-label identifies the FORMAT statement, but is not an executable statement label (in other words, you can't GOTO a format-label). Since format-labels are not the same as executable statement-labels, you can use the same name in each section of the program (e.g., you could have a format-label of PRINTOUT and a executable statement-label of PRINTOUT in the same program).

Here's a sample FORMAT statement:

```
HEADING: FORMAT "Day",@(0);"Amount",@(25)
```

Note: A FORMAT statement is an inert object in an IB program. You "bring it to life" by using the appropriate input/output statement. For example:

```
PRINT (1,HEADING)
```

As you can see, this is an alternate version of the PRINT statement. Instead of including print items on the same line as the PRINT statement, this PRINT refers to a FORMAT statement that contains those items.

Here's the FORMAT statement for the detail lines in our sample program:

```
DETAIL: FORMAT I,@(0);AMOUNT,@(10),"##,###,###,###,##0.00-"
```

Notice that the edit mask is the third parameter in the print item for the AMOUNT variable.

The associated PRINT statement is:

```
PRINT (1,DETAIL)
```

Take a look at the sample program named SAMPLE8 (source name is SAMPLE8.MTB) and you'll see how the edit masks makes the large numbers look better.

Summary

While this chapter is short, it introduces some very important topics, including printer names, logical-unit-numbers, opening and closing a printer, dealing with runtime exceptions, printing to a printer (and virtual printer), using FORMAT statements, and edit masks. For the sake of simplicity, some details are missing from this chapter, but we'll get to them soon enough.

In the next chapter, we'll take a look at data files.