Chapter 8 – Functions

This chapter is about IB functions. Some functions are simple (stripping blanks from a string), while others are somewhat complex (moving a specified number of characters from one string into another string, starting at a specified location in the second string). Other functions return numerous bytes of status information, such as file status, device status, partition status, or logical unit number status. The list goes on.

The IB functions are presented in categories. For additional information on any of these functions, please see the reference documentation at the IB web site (www.internetbasic.com).

String stripping functions

The STRIP function strips leading and trailing blanks from a string. There are two related functions, STRIPL (which strips only the leading blanks) and STRIPR (which strips only the trailing blanks). The syntax is:

```
STRIP(string-argument)
STRIPL(string-argument)
STRIPR(string-argument)
```

When a string is stripped, its current length is adjusted accordingly. The current length is a numeric value that represents the number of characters of data stored in the string field.

String length function

The LEN function returns the current length of a string-argument. This value is very helpful for programs that need to determine how many characters of data are stored in a string field. (Note: It returns the current length, not the defined length.) The syntax is:

```
LEN(string-argument)
```

String adjustment functions

The ADJUSTL and ADJUSTR functions left-justify and right-justify a string-argument, respectively. For example, if the ADJUSTL function is used on a string that contains leading blanks, those blanks are removed and the remaining data is moved so that the first character of data appears in the first position of the string. The ADJUSTR function eliminates trailing blanks and moves the data so that the last character appears in the last position of the string. These functions to not affect the current length of the string. The syntax is:

```
ADJUSTL(string-argument)
ADJUSTR(string-argument)
```

Case adjustment functions

The UCASE function converts lower case characters in a string-argument to upper case characters. Other characters in the string-argument (e.g., upper case characters, special characters) remain unchanged. The LCASE function converts upper case characters to lower case. The syntax is:

```
UCASE(string-argument)
LCASE(string-argument)
```

Substring functions

SUB

The SUB function returns a substring of a specified string-argument. The syntax is:

```
SUB(string-argument,starting-position,length)
```

For example, the following statement returns the 5th, 6th, and 7th characters of the data stored in A$:

```
B$ = SUB(A$,5,3)  ! starting position=5, length of substring=3
```

RSUB

The RSUB function returns a substring from the right-hand side of a string argument. The syntax is:

```
RSUB(string-argument,starting-position,length)
```

The starting-position represents value counting from the right-hand side of the string. This function uses the current length (not the defined length) when determining the right-hand-most character.

For example, the following statement returns 5 characters from the right-hand size of the ADDRESS$ field:

```
ZIP$=RSUB(ADDRESS$,1,5)
```

MOVE

The MOVE statement moves a specified number of characters from one string into another string, starting at a specified location in the second string. The syntax is:

```
MOVE(string-argument-1,string-argument-2,starting-position,length)
```

Note: This function does not return a value; it simply moves a portion of the first string into the second one.

For example:

```
MOVE(A$,B$,2,3)
```

If A$ = “ABCDEFG” and B$ = “123456789”, after the move operation, and B$ is “1**ABC**56789”.

String position functions

POS

The POS function returns the position of one string argument within a second string argument. The syntax is:

```
POS(string-1,string-2)
```

For example:

```
N = POS("SMITH",NAME$)
```

In this example, the string constant "SMITH" is compared to the string variable NAME$. If "SMITH" is located anywhere within NAME$, its position is returned and stored in the numeric variable N.

If "SMITH" is not found in the variable NAME$, the POS function returns a value of 0. This programming technique can be useful for determining if a particular string value is contained within a specific data field.

RPOS

The RPOS function returns the position of one string argument within a second string argument **scanning from right to left**. The syntax is:

```
RPOS(string-1,string-2)
```

For example, assume that A$ = “c:\folder1\misc”:

```
N = RPOS("\",A$)
```

The RPOS function scans A$ from the right-hand side, looking for the position of the first backslash. It finds one at position 11 (counting from the left-hand side), which is the value that is assigned to N.

String swapping function

The SWAP function exchanges the values of two string variables. The syntax is:

```
SWAP(string-varaible-1,string-cariable-2)
```

Note: This function does not return a value; it simply exchanges the two strings.

String character duplicator function

The STRING function assembles a string of identical characters of specified length. The syntax is:

```
STRING(string-argument,numeric-argument)
```

For example, the following statement assembles a string of 50 asterisks:

```
A$ = STRING("*",50)
```

Keyed file functions

The following functions return the first and last keys, respectively, in a keyed file:

```
FIRST(logical-unit-number [,EXCP=statement-label])
LAST(logical-unit-number [,EXCP=statement-label])
```

The KEY function returns the value of the next key, based on the position of the file pointer. The syntax is:

```
KEY(logical-unit-number [,EXCP=statement-label])
```

The PREV function returns the value of the previous key. The syntax is:

```
PREV(logical-unit-number [,EXCP=statement-label])
```

The RECNUM function returns the record number (numeric) of the keyed file that is open on the specified logical unit number. The syntax is:

```
RECNUM(logical-unit-number [,EXCP=statement-label])
```

Path function

The PATH function returns the full path associated with the specified IB directory label. The syntax is:

```
PATH(directory-label)
```

For example:

```
A$ = PATH("BAS")
```

This statement returns the full path associated with the IB directory named “BAS.”

Encryption functions

The ENCRYPT function encrypts a string value by using a specified seed string. The syntax is:

```
ENCRYPT(plaintext-string-value,seed-string)
```

The seed-string may contain up to 254 characters. The longer the seed-string, the better the encryption.

For example:

```
A$ = "Here is some plain text."
SEED$ = "QWERTY123456ASDFGH567890"
B$ = ENCRYPT(A$,SEED$)
```

The B$ variable contains the encrypted version of A$. To decrypt this value, use the DECRYPT function and the same seed-string that was used to encrypt the plain text. The syntax is:

```
DECRYPT(encrypted-text,seed-string)
```

For example:

```
SEED$ = "QWERTY123456ASDFGH567890"
A$ = ENCRYPT(B$,SEED$)
```

Encoding functions

The BASE64 function performs a standardized encoding of a string-argument into printable characters, regardless of the value of the string-argument. The syntax is:

```
BASE64(string-argument,numeric-error-argument)
```

The numeric-error-argument is a numeric variable with a length and precision of at least 1.0. If the BASE64 function successfully encodes the string-argument, the numeric-error-argument is set to 0. If the conversion is unsuccessful (e.g., if the encoding overflows the string accumulator), it is set to 1.

This function allows hex strings to be encoded and used for such purposes as hidden fields in an HTML form, cookies, etc. For more information about Base 64 encoding, please see RFC 1421.

Note: For every 3 characters of 8 bits each in the string-argument, the BASE64 function returns 4 characters of 6 bits each. This means that the resulting string is longer than the string-argument. For example, if the string-argument is 30 characters, the result is 40 characters.

To de-encode a Base 64 string, use the BASE256 function. The syntax is:

```
BASE256(string-argument,numeric-error-argument)
```

The BASE256 function de-encodes the string-argument (a Base 64 string) into its original value.

The numeric-error-argument is a numeric variable with a length and precision of at least 1.0. If the BASE256 function successfully de-encodes the string-argument, the numeric-error-argument is set to 0. If the conversion is unsuccessful (e.g., if the string-argument contains an invalid Base 64 character), it is set to 1.

Note: For every 4 characters in the string-argument, the BASE256 function returns 3 characters.

Numeric functions

ABS

The ABS function returns the absolute value of a numeric-argument. The syntax is:

```
ABS(numeric-argument)
```

FPT

The FPT function returns the fractional portion of a numeric-argument. The syntax is:

```
FPT(numeric-argument)
```

INT

The INT function returns the integer portion of a numeric-argument. . The syntax is:

```
INT(numeric-argument)
```

SQRT

The SQRT function returns the square root of a numeric-argument. The syntax is:

```
SQRT(numeric-argument)
```

SGN

The SGN function returns the sign (positive, negative or zero) of a numeric value according to the following chart:

Numeric-argument	Value returned
Positive	1

Zero	0
Negative	1-

The syntax is:

```
SGN(numeric-argument)
```

RND

The RND function returns a pseudo-random value less than 1 but greater than or equal to 0. The returned value has a length/precision of 8.8 (i.e., all 8 digits are to the right of the decimal point).

RND(0) generates a pseudo-random number, while using a non-zero argument, such as RND(1), randomized the value returned by this function.

The syntax is:

```
RND(numeric-argument)
```

Conversion functions

IB supports several functions that convert data from one form to another.

STR

The STR function converts a numeric argument to a string value. The syntax is:

```
STR(numeric-argument)
```

For non-expressions, the length of the result is the length of the numeric-argument plus 1 character to account for the sign of an integer, or plus 2 characters to account for the sign and decimal point of a real number.

For expressions, the STR function returns 18 characters for a real number (16 digits plus a decimal point and sign) or 17 characters (16 digits plus a sign for an integer). The "precision" in the resulting string is determined by the most precise element in the expression.

The STR function provides a sign character at the right-hand side of the converted value. If the numeric-argument contains fractional digits, the STR function inserts a decimal point in the correct location.

NUM

The NUM function converts a string-argument into its equivalent numeric value, only if the contents of the string-argument are numeric characters. The syntax is:

```
NUM(string-argument,numeric-error-argument)
```

The numeric-error-argument is a numeric variable or numeric array element with a length and precision of at least 1.0 If the NUM function successfully converts the string-argument, the numeric-error-argument is set to 0. If the conversion is unsuccessful, it is set to 1.

ASC

The ASC function returns the decimal value of the first byte of the string-argument using the 8-bit ASCII code. The syntax is:

```
ASC(string-argument)
```

CHR

The CHR function returns a single-byte ASCII character corresponding to the 8-bit binary number equivalent of the numeric-argument. The syntax is:

```
CHR(numeric-argument)
```

The numeric-argument is a decimal value between 0 and 255, inclusive. Only the absolute integer portion of the numeric argument is used for the conversion; the sign and fractional digits (if any) of the numeric argument are ignored.

BINARY

The BINARY function returns 8 bytes for each character in the string-argument. The returned bytes represent the 8 bits in each character of the string-argument. If the bit is "on," the corresponding byte will be "1." If the bit is "off," the corresponding byte will be "0." The syntax is:

```
BINARY(string-argument)
```

For example:

```
A$ = "COOL"
B$ = BINARY(A$)
```

B$ is the binary equivalent of "COOL" – which is:

```
01000011010011110100111101001100
```

ASCHEX

The ASCHEX function converts an ASCII hex string to its equivalent ASCII value. The syntax is:

```
ASCHEX(string-argument, numeric-error-argument)
```

The string-argument represents the string to be converted. Each two bytes of this string-argument must represent one ASCII character (this function provides the conversion to ASCII). The string-argument may be a string constant, a single-element string variable, a string array element, a string expression, or a string function.

The numeric-error-argument reports the success or failure of the conversion. It is set to 1 if there is a conversion error; otherwise, it is set to 0. The numeric-error-argument must be a single-element numeric variable or a numeric array element.

If there is a conversion error (i.e., if the string-argument contains non-hex characters), the ASCHEX function returns a null result.

HEXASC

The HEXASC function converts an ASCII hex string to its ASCII hex equivalent, character for character. The syntax is:

```
HEXASC(string-argument)
```

The string-argument represents the string to be converted. The string-argument may be a string constant, a single-element string variable, a string array element, a string expression, or a string function.

The result of the HEXASC function contains twice as many characters as the string-argument (because each byte of the string-argument will be represented by 2 bytes in ASCII).

DECHEX

The DECHEX function converts a decimal number to its hexadecimal string equivalent. The syntax is:

```
DECHEX(numeric-argument, numeric-error-argument)
```

The numeric-argument represents the value to be converted. It may be a constant or variable. The value of the numeric-argument may not exceed 65,535 (i.e., hex "@FFFF@"). The numeric-argument may be a numeric constant, a single-element numeric variable, a numeric array element, a numeric expression, or a numeric function.

The numeric-error-argument reports the success or failure of the conversion. It is set to 1 if there is a conversion error; otherwise, it is set to 0. The numeric-error-argument must be a single-element numeric variable or a numeric array element.

HEXDEC

The HEXDEC function converts a two-byte binary string to its decimal equivalent. The syntax is:

```
HEXDEC(string-argument)
```

The string-argument represents the two-byte binary string to be converted. The string-argument may be a string constant, a single-element string variable, a string array element, a string expression, or a string function.

The maximum possible value returned by the HEXDEC function is 65,535, the decimal equivalent of "@FFFF@" hex.

INTEL and INTELD

The INTEL function returns a 2-byte string in Intel (byte-reversed) format representing the hex equivalent of the decimal numeric-argument. The syntax is:

```
INTEL(numeric-argument)
```

The INTELD function works just like the INTEL function except that it returns a double-word (4-byte) string. The syntax is:

```
INTELD(numeric-argument)
```

The INTEL and INTELD functions were implemented in support of the IB Windows Driver Library. Both functions can be used in data section of the program (with a constant numeric argument) and in the code section of the program (with a constant or variable numeric argument).

Date functions

DATETONUM
DATE2NUM
NUMTODATE
NUM2DATE

Status functions

STS
PSTAT
DSTAT
FSTAT

Special purpose functions

CHKSUM

MID