Chapter 6 – Dealing with Runtime Exceptions

By now you have a good idea of how InternetBasic programs are organized and how they store and retrieve data. By studying the sample programs in the previous chapters, you've also seen some of the ways that IB programs deal with runtime exceptions. This chapter provides you with a better understanding of the events that cause runtime exceptions and the steps your program can take to recover from them.

To review, a runtime exception occurs when a program executes an instruction that results in an unexpected or extraordinary outcome. Here are some examples:

- A program attempts to open a printer while the printer is being used by another IB program/user. (*Exception: device unavailable*.)
- A program attempts to open a data file, but the file doesn't exist. *(Exception: file not found.)*
- A program attempts to read a record from a keyed file, but the key does not exist. *(Exception: key not found.)*
- A program attempts to read a record from a keyed file, but the record is locked (extracted) by another IB program/user. *(Exception: record unavailable--extract error.)*
- While reading a data record, a program attempts to read a numeric value, but the actual data field contains a non-numeric character. *(Exception: non-numeric input in a numeric field.)*
- While reading through a data file in sequential (or key-sequential) order, a program reaches the last record in the file, but attempts to read one more record (or key). *(Exception: end-of-file.)*

As you can see, runtime exceptions typically occur when input/output statements are executed.

Rather than stop a program when an exception occurs, the IB runtime system informs it about the exception and provides ways for it to recover. Information is available via a system variable named EXCP. This variable contains a number that corresponds to the exception that occurred. For example, when an end-of-file exception happens, EXCP is set to 2. When a key-not-found exception occurs, EXCP is set to 32. All of the IB runtime exception codes are described below.

To recover from exceptions, an IB program can branch to a specific statement label when an exception happens (in other words, jump to some location in the code that is out of sequence from where the program was when the exception occurred). Such locations mark the beginning of "exception routines." By testing the value of the EXCP system variable, an exception routine can determine what exception occurred and then decide when action to take (such as displaying a message for the user, finding the next matching record, etc.). Some of these techniques have already been demonstrated in the sample programs.

Of course, the program does not have to include exception routines, in which case a runtime exception will cause the program to halt (and will also cause the IB runtime exception reporting program to run).

There are several ways to include exception routine branching instructions in IB programs. The most common is:

```
statement, EXCP=statement-label
```

Where the EXCP= parameter provides the statement-label where this program branches if an exception occurs when this statement is executed. (If no exception occurs, the program does not branch anywhere; it simply processes the next instruction.)

An alternative is to have the program branch to a subroutine when an exception occurs, which is written like this:

```
statement, EXCPSUB=statement-label
```

Another alternative is to set a default exception path, one that is used if no specific EXCP= or EXCPSUB= parameter is included for a particular statement. This is written as follows:

```
ERRORTO statement-label
```

This statement has an alternative form, which is:

```
ERRORTO SYSTEM
```

This version of ERRORTO clears the previous ERRORTO statement-label that was in effect.

Finally, the default exception path can be a subroutine, in which case the statement is:

```
ERRORSUB statement-label
```

And, as you might have expected, the way to clear the previously set ERRORSUB label is:

```
ERRORSUB SYSTEM
```

In an exception routine, there are several ways to decide what action to take. We've already mentioned the EXCP system variable, but here are two other statements that are useful. First, here's a statement that jumps back to the instruction where the exception occurred:

```
AGAIN
```

Take care when using the AGAIN statement, as it could result in a program getting stuck in an endless loop (assuming that the offending statement keeps generating the same runtime exception time after time).

Second, after testing for the anticipated exceptions, a program may want to deal with unanticipated ones. The following statement, which typically appears at the logical ending of an exception routine, halts your program and runs the IB runtime exception reporting program:

```
ERROR
```

The exception reporting program provides information about the exception and where it occurred. This tool also provides a way for you to create a log of all IB runtime exceptions, which can serve as a valuable diagnostic tool for system managers. (See Chapter ??? for more information.)

At the risk of being redundant, here's a summary of the exception processing options:

Trapping exceptions

- statement, EXCP=statement-label
- statement, EXCPSUB=statement-label
- ERRORTO statement-label
- ERRORTO SYSTEM
- ERRORSUB statement-label
- ERRORSUB SYSTEM

Handling exceptions

- Test the value of the EXCP system variable
- Use the AGAIN statement to branch back to the instruction that caused the exception
- Use the ERROR statement to halt the current program and run the exception reporting program

As previously mentioned, the value of the EXCP system variable equates to a specific runtime exception. Here is a list of the EXCP values and a description of each exception. Values less than 70 are considered to be non-fatal exceptions (i.e., they can be trapped by an IB program).

EXCP	Description of exception
0	No exception reported
1	Not in Current Use
2	End of file This exception happens when a READ, INQUIRE, or EXTRACT statement attempts to move the file pointer past the last record (text or sequential file) or last key (keyed file). This is the normal consequence or reading through a file

	in sequential or key-sequential order until the program reaches the last record. If the program tries to read one more record (key), this exception occurs. This exception also happens if the PREV function is executed when the pointer is at the first key in a keyed file (i.e., the program attempts to retrieve a key prior to the first key).
3	Unable to Allocate Space for File
4	Keyed File Write Without Index This exception happens when a WRITE statement attempts to write a record to a keyed file without specifying a key value.
5	Create Without Directory Label This exception happens when a CREATE statement attempts to create a text, sequential, or keyed file without specifying a directory label.
6	Directory Not Found on ACCESS This exception happens when the ACCESS statement attempts to access an IB directory than does not exist.
7	File Unavailable--Lock/Unlock Error This exception happens when an OPEN statement attempts to open a text, sequential, or keyed file that has been opened and locked by another IB program/user.
8, 9	Not in Current Use
10	File not available
11	File Not Found This exception happens when an OPEN statement attempts to open a text, sequential, or keyed file and the specified filename is not found. This is mostly likely doe to the fact that the named file does not exist, or that the file cannot be found among the configured and accessed IB directories.
12	File Already Exists This exception happens when a CREATE statement attempts to create a text, sequential, or keyed file and the specified filename already exists.
13	DOS file already exists
14-23	Not in Current Use
24	Unable to Rename Disk to Disk
25	Invalid DOS Call
26	Invalid Message Call
27	DOS Returned an Error
28	Invalid Priority Call
29	Communications line error
30	Device Inoperative

	This exception happens when an OPEN statement attempts to open a device (such as a printer) that is not operative.
31	Device or Directory Unavailable This exception happens when an OPEN statement attempts to open a device or directory that is unavailable.
32	Key Not Found This exception happens when a READ, INQUIRE, or EXTRACT statement attempts to retrieve a key/record from a keyed file and the specified key is not found in the key tree. The exception also happens when a REWRITE statement attempts to rewrite a key/record to a keyed file and the specified key does not exist. When using REWRITE, only existing keys may be used.
33	Record Unavailable--Extract Error This exception happens when a READ or EXTRACT statement attempts to retrieve a key/record from a keyed file and the specified key/record is locked (extracted) by another IB program/user. (Note: This exception does not happen with the INQUIRE statement.)
34	Logical Unit # Unavailable for Open This exception happens when an OPEN statement attempts to open a file or device using a logical unit number that has previously been opened.
35	File Not Open for Read or Write This exception happens when a file-oriented input/output statement attempts to read or write to a text, sequential, or keyed file that is not open.
36	Can't Perform Function on Open File This exception happens when a program attempts to perform a function on an open file that is not allowed (e.g., trying to rename an open file).
37	Not in Current Use
38	Device or Target File System Unable to Perform Function
39	Not in Current Use
40	Invalid Window Specified
41	Edit Mask Length Incorrect This exception happens when the length of an edit mask in a FORMAT statement does not match the length/precision of the numeric value being printed.
42	Not in Current Use
43	File extension not allowed for this operation
44	I/O Buffer Overflow

	This exception happens when a read/write operation is attempted with too many bytes, thus exceeding the input/output buffer size.
45	Invalid Parameter
46	Non-Numeric Input in a Numeric Field This exception happens when an input operation (such as READ, INQUIRE, or EXTRACT) attempts to retrieve a non-numeric character using a numeric variable. This usually indicates a problem with the FORMAT statement (e.g., variables in the wrong order, variables specified with the wrong length/precision), or a problem with the data being retrieved (e.g., fields were written in the wrong order, or written with the wrong length/precision).
47	Parameter Too Large
48	Invalid Key or Record Size: CREATE This exception happens when the CREATE statement attempts to create a file using an invalid key size or record size.
49	Invalid Logical Unit Number This exception happens when the program specifies an invalid logical unit number. Valid logical unit numbers range from 0 through 49.
50	Array Subscript Out of Range This exception happens when a subscript of an array is specified beyond the boundaries of the array.
51, 52	Not in Current Use
53	Invalid File Access This exception happens when a file is access using an invalid method (e.g., specifying a key for a text file).
54	Invalid Function for Named File
55	Not in Current Use
56	Record Already Exists for INSERT This exception happens if an INSERT statement attempts to insert a duplicate key/record to a keyed file. When using INSERT, only new key/records are allowed.
57-59	Not in Current Use
60	Cannot IPL while other Users Active This exception happens if a user attempts to stop the IB runtime system while at least one other IB program is still running.
61	Program Not Found The exception happens when a RUN, ENTER, or ACTIVATE statement attempts to execute a program that is not found in the IB directories that are currently accessed.

62	Program too large
63	Cannot Run Non-Object File The exception happens when a RUN, ENTER, or ACTIVATE statement attempts to execute a file that is not an IB object program.
64	Invalid Partition Name This exception happens when a statement or function refers to a partition name that is invalid.
65	Partition Busy, Program Running This exception happens when an ACTIVATE statement attempts to execute an IB program is a partition where one is already executing.
66	Performed an EXIT when ENTERLEVEL value was 0 This exception happens when the EXIT statement is attempted when the ENTERLEVEL system variable equals 0, which indicates that the current program is not executing as a subprogram. See Chapter ??? for more information about IB subprograms.
67	Partition Busy, Normal Termination
68	Terminate Complete, Task Wasn't Idle
69	Activate Unsuccessful/Task had Error Pending

Values of EXCP that are greater than or equal to 70 are considered to be fatal exceptions. An IB program cannot trap these exceptions. When they occur, the IB program halts and the exception reporting program takes over, displaying (and optionally, logging) the fatal exception.

EXCP	Fatal exception
70	ESCAPETO/SUB not active for INTERRUPT
71	DOS Failed to Transfer Read Data
72	IRQ 3 Failure from Comet Board
73	Illegal Jump to Location Zero
74	Bad function call to moveable segment
75	Resources Left Allocated After Call
76	File Size Limit Reached in Demo Mode
77	Attempted Write with Zero Length
78	Error Performing ENTER/EXIT Program
79	Key Block Offset Too Large
7A	Attempt to EXTRACT Keyed Only File
7B	Maximum File Size Exceeded
80	Maximum I/O Buffer Size Exceeded
81	Printer I/O Error
82	Top Key Sector not in Delete Stack
83	DOS Disc Error

84	Key Sector Search Level Impossible
85	Invalid Instruction/Real Parameters
86	File Descriptor List Overflow
87	Unexpected EOF from DOS
88	Keys but NO Records in File
89	Unsupported Reference
8D	503 Spool Jobs File Problem - CLEAR Spool and retry.
8E	Node Lock Timed Out
90	Disk Full -- DOS Wrote Less Than Expected
91	Invalid/Unsupported System Call
92	GOSUB Stack Underflow
93	GOSUB Stack Overflow
94	Directory Entry Unavailable / CLOSE
95	Next Key Unavailable for Deletion
96	New Key Found during INSERT
97	New First Key found during INSERT
98	Extracted Record not found in Table
99	Error Received from DOS
9A	NLM Error in key tree
9B	NLM Transport Error (SEND)
9C	CometServe ENTER error
9D	CometServe EXIT error
9E	NLM file system discovered corruption in an I00 file
9F	CometServe bad function request

Summary

This chapter serves as reference material for dealing with runtime exceptions. As you have seen, InternetBasic provides a very easy way to identify and handle these situations. As you write more and more IB programs, you will no doubt develop a personal style for your own exception routines. Keep this chapter handy for those times.

In the next chapter, we'll take a formal look at the control structures in IB, many of which you've already seen in the sample programs.