

# **Writing Applications to Automate Host Interactions**



**Hummingbird**

## Writing Applications to Automate Host Interactions

The following is a primer to help developers write applications that automate their way through host applications. Synchronizing is the hardest part of any automation because it is dependent on how the host application was written. Consider the following scenarios and you will quickly realize that synchronization is sometimes not so obvious:

- 1) You are at the TSO Ready prompt and you type 'ISPF'. The host responds with one 3270 I/O which displays the ISPF panel and unlocks the 3270 keyboard.
- 2) You are at the TSO Ready prompt and you issue a command such as LISTCAT. The host responds with multiple 3270 I/Os and the last I/O unlocks the 3270 buffer.
- 3) You are at a CICS application screen and you issue some type of query which may take some time. The host responds immediately with one I/O which states 'Query in Progress' and unlocks the 3270 keyboard. Now, you wait until you see the response come back, something which may take 100 ms or 5 seconds.

In cases (1) and (2), the synchronization is simple. You press an action key and simply wait for the host to unlock the 3270 keyboard and you can proceed with the next request.

In case (3), the 3270 keyboard is unlocked before the transaction is complete. Therefore, you must wait for some additional visual indicator to know that the transaction is complete (either good completion or some kind of failure).

Therefore, when synchronizing with host applications, it is always recommended to perform two types of wait after every action key (PFx, PAX, Enter, Clear). The first is to wait for the 3270 keyboard to be unlocked by the host system. This generally means that the host is now ready to accept more input. The second check is to wait for a well known string which appears as a result of the request. Do not wait for a string which is already present on the screen.

The code below uses HostExplorer's OLE Automation interface which is the most common API used to automate host applications. It can be called from Hummingbird Basic, Visual Basic, C/C++, Delphi, etc. The code assumes that the 'HE' object points to the HostExplorer root object which is commonly set using 'Set HE = CreateObject( "HostExplorer" )'. The 'Host' object points to the session we are automating. This can be 'CurrentHost' or a specific host object.

The sample sequence below is used to automate one interaction:

```
iPSWaitTime% = 60
```

```

' Enter a command and press ENTER
Host.Keys "LOGON PIERRE@E"

' Wait for the host to update the host screen and unlock
' the 3270 keyboard.
iRc% = Host.WaitPSUpdated( iPSWaitTime%, TRUE )

' Check to see if the wait finished successfully or if it timed out
If iRc% <> 0 Then
    MsgBox "Host did not respond within update period"
    ' Goto error handler...
End If

' Now, wait for a well known string to appear. In this sample,
' we will wait for that string to appear anywhere on the screen
iRc% = Host.WaitForStringRC( "Password ==>", 0, 0, 0, iPSWaitTime%,
FALSE )

If iRc% = 0 Then
    MsgBox "String was not found within timeout period"
    ' Goto error handler...
End If

```

NOTE: The WaitPSUpdated call as shown above has an additional parameter which is not documented in the online help for 8.0 or 9.0. This parameter is important for automation. When set to TRUE (Default is FALSE), this instructs the WaitPSUpdated to wait for the host to also unlock the keyboard before returning. Therefore, the logic for this call will be the following:

- a) Wait for the host to update the screen within the timeout period
- b) (3270/5250 Only) If the screen is updated, wait for the host to unlock the keyboard within the timeout period.
- c) (3270/5250 Only) If the keyboard is unlocked, wait an additional 300 ms to allow host to accept input.

There are many enhancements that are required to make this code sample 100% reliable.

First, you need some error handling. What happens if one of the waits times out? You probably need to check to see if the session is still connected and then check for possible error strings returned from the host. Some programmers prompt the user to see if the user wants to wait for an additional period of time. Next, the wait for string is generic. It waits for the string to appear anywhere on the screen.

If many host applications, especially those which are transaction based, there is typically a panel identifier somewhere on the screen, usually in a fixed location. Waiting for this panel identifier is an excellent method to synchronize with the host application. You can also use that identifier to handle error conditions. Maybe you will get back a panel different than expected. The identifier will make your job easy.

In order to make your application fast and efficient, you may need to enter large amounts of data on the host screen. There are several ways to do this in the OLE API and in HLLAPI. In OLE, the following methods can be used:

- a) Host.Keys
- b) Host.RunCmd
- c) Host.PutText
- d) Host.Fields(x).Text

The first option (a) is commonly used but is not efficient for bulk I/O because it actually moves the cursor as it enters data. This method is identical to someone typing data on the keyboard. This method is useful mainly if you want the cursor to move as if the data was entered by a real person. This method respects all keyboard states and modifiers such as Insert and Entry Assist (WordWrap).

The second option (b) is only useful for sending system commands or action keys.

The third option (c) is the most efficient because it puts the data directly into the 3270 buffers without moving the cursor. It is not affected by Insert or Entry Assist.

The last option (d) is used to enter data into fields by getting a field ID and using the Field object or Fields Collection. This option is also very efficient and nice because it creates a geometry independent input method.

The last important item in creating fast and efficient automation applications is scraping data from the host screen. Again, there are several methods which can be used to get data from the screen.

- a) Host.Text
- b) Host.TextRC
- c) Host.Row
- d) Host.Fields(x).Text

The first option (a) returns the entire screen as one long text string. All the rows are appended and the screen is represented as one long string. You can use string handling functions such as Left\$, Mid\$, Right\$ to extract various parts in Basic.

The second option (b) returns a portion of the screen from the given location. See the online help for all the options for this method. This is a very powerful extraction tool for automation.

The third option (c) returns a given row from the host screen.

The last option (d) returns the text of a specified field.

While creating your automation application, try not to create loops where you wait for certain events to happen. These will drive the CPU to 100% and are not efficient. Use the wait functions in the emulator to wait for events.