How to Use HLLAPI Functions to Program Wait Times

Attachmate Technical Bulletin #195

Contents

Purpose

This document defines various HLLAPI (High-Level Language Application Programming Interface) programming wait functions, which allow for more efficient synchronization in between the timing of HLLAPI and host applications.

HLLAPI application developers often look for an easy way to determine when the host is ready to process more information so that they can write programs to continue as soon as the host is ready to receive more information.

Often a wait time is hard-coded a specified number of seconds to be sure that the host is ready for input for all situations. But for application developers who are writing applications to run on different hosts with different host response times, it is important that code written for one environment runs in all environments.

Use any of the following HLLAPI function calls to make programmed wait calls more robust in all environments.

Functions

NOTE: The following solutions are only suggestions. Results will vary.

Function 4 (Wait) in a loop

A Function 4 identifies when an X() clock appears in the OIA (Operator Information Area) and waits until the X() clock goes away before processing the next command. However, it is possible for the X() clock to go away before the host is finished sending all of the information packets, prompting the HLLAPI program to send information to the host when the host is not ready. This causes an input inhibit condition to occur on the host side (X-I or X-f in the OIA).

To workaround this issue, account for the fact that the host will be sending more packets of information with the X() clock blinking on and off by putting a Function 4 in a loop with a wait time-out. Write the program to move on to the next command if the X() clock does not appear for more than five seconds.

Function 6 (Search Presentation Space) with time-out

A Function 6 by itself is usually sufficient to provide the information needed before continuing with the code. The program looks at the host screen for a specific string in the host session. Write the application to send information to that presentation space when the string is found.

Sometimes Function 6 cannot prevent inhibit conditions when code is written to send information to the host session immediately following a Function 6 found string. It may be necessary to perform searches of host presentation spaces, and then wait about two seconds in case the host is not ready for user input.

Function 7 (Query Cursor Location) in a loop

The host cursor usually moves erratically during host screen updates, but it is possible in a loop to query where the cursor is. As soon as the cursor stays in one position, it can be safely assumed that the host is ready to receive more information and process the next command. Function 7 may be the best, yet most unused, way to wait for the host to be ready for more input.

Function 13 (Copy OIA) for X() clock in data string

A Function 13 returns host status information in its data string return bytes. This includes the 4, the B, the block, XPROG errors, and the X() clock. In a manner similar to the Function 4 (Wait) solution, read the returning data string information and look for an X() clock. If the X() clock goes away, loop until the X() clock does not come back for more than three or five seconds, depending on the host response time.

Function 13 (Copy OIA) group bits

A Function 13 has 103 bytes of data string return bytes. Bytes 82 through 103 are

called the OIA bit groups. Each byte returns a bit number that defines a particular OIA characteristic. Group 8 (bytes 89 through 93) is very useful as it describes why input is inhibited in the current presentation space

Function 23 (Start Host Notification) and Function 24 (Query Host Update) in a loop

Function 23 and Function 24 are designed to be used together to determine if the host session has experienced an update of the presentation space, the OIA, or both. The user does not have to be connected to the presentation space to start host notification in a session, so host notification can be started for all host sessions. If an update occurs, use Function 24 to see which session was affected.

Function 23 (Start Host Notification), Function 18 (Pause), and Function 24 (Query Host Update)

This combination is very similar to the Function 23 and Function 24 pair, except that Function 18 allows the program to wait until a host update occurs. This requires setting the IPAUSE parameter in HLLAPI Function 9 (Set Session Parameters). Once the parameter is set, call Function 18 to wait until a host update occurs in a session in which the host notification has been started.

When the update is complete, a return code of 26 occurs from Function 18. Call Function 24 to see whether the presentation space, the OIA, or both, are updated in the session being monitored.

Function 4 (Wait), Function 6 (Search Presentation Space), and Function 7 (Query Cursor Location) in a loop

This is an example of how several functions can be used in conjunction to wait for the host to be ready to receive more input. For example, use the loop to look for an X() clock, to wait for the X() clock to go away, and then to look for a specific host string in the host session. If the string is not present, loop on Function 4 again.

Example

The following example is a robust HLLAPI wait routine written in C. It employs a combination of HLLAPI functions to ensure that the host system is quiet for a specified length of time. This is the type of routine that should be called every time an aid key is sent to the host using Function 3 (Send Key).

cSessID	The session short-name of the current HLLAPI-connected session.

dwSettleTime	The amount of time the OIA and presentation space must be stable before this function is allowed to return success (no presentation space updates and no OIA X() clock flickers). Designate at least a second (1000 milliseconds) for this.
dwTimeOut	The total time allowed for waiting before the function returns with an error, often 20 seconds (20000 milliseconds) or more.
acs3ehap.h	The header file included with EXTRA! products that defines the constants used.

```
int HLLAPIWaitForQuiet (char cSessID, unsigned long dwSettleTime,
                        unsigned long dwTimeOut)
{
     int wReturn;
     unsigned long SettleTimeStart = 0,
                   TimeOutStart    = 0;
     int   Function,
           wStrLen,
           wRetCode;
    char far      String[256];

     TimeOutStart = GetTickCount(); //Win32 API function

       // call Set Session Parameters for NWAIT and IPAUSE
     Function = SETSESSION;
     strcpy (String, "NWAIT,IPAUSE");
     wStrLen = strlen(String);
     wRetCode = 0;
     HLLAPI (&Function;, String, &wStrLen;, &wRetCode;);

       // Call Start Host Notify
     Function = STARTHOSTNOTIFY;
     strcpy (String, " B    ");
     String[0] = cSessID;
     wStrLen = 255;
     wRetCode = 0;
     HLLAPI (&Function;, String, &wStrLen;, &wRetCode;);

     while (TRUE)
     {
           // Call Wait
          Function = WAIT;
          String[0] = 0;
          wStrLen = 0;
          wRetCode = 0;
          HLLAPI (&Function;, String, &wStrLen;, &wRetCode;);
```

```
     if (wRetCode ≠ 0)       // X-Clock not clear
     {
          SettleTimeStart = 0;   // clear the settletime

          // check for timeout
          if (GetTickCount() - TimeOutStart ≥ dwTimeOut )
          {
               wReturn = wRetCode;
               break;
          }
     }
     else
     {
          if (SettleTimeStart == 0)    // begin counting the settletime
          {
               SettleTimeStart = GetTickCount();
          }

          if (GetTickCount() - SettleTimeStart ≥ dwSettleTime)
          {
               wReturn = wRetCode;
               break;
          }
     }

     // pause for interuptable 1/2 second
     Function = PAUSE;
     String[0] = 0;
     wStrLen = 1;
     wRetCode = 0;
     HLLAPI (&Function;, String, &wStrLen;, &wRetCode;);


     // call QueryHostUpdate to clear the notification
     Function = QUERYHOSTUPDATE;
     String[0] = cSessID;
     String[1] = 0;
     wStrLen = 1;
     wRetCode = 0;
     HLLAPI (&Function;, String, &wStrLen;, &wRetCode;);
}//while

// call STOPHOSTNOTIFY
Function = STOPHOSTNOTIFY;
String[0] = cSessID;
String[1] = 0;
wStrLen = 0;
wRetCode = 0;
```

```
     HLLAPI (&Function;, String, &wStrLen;, &wRetCode;);

     return wReturn;
}
```

For additional information, refer to the EXTRA! Developer Series API SDK (Software Development Kit).

Bulletin Date: September 1999

KBCode: GTOOL