The Language Monitor

The Language Monitor is part of the Windows driver and is located between the Driver UI and the Port Monitor, which takes care of the direct communication with the selected port.
The Zebra Language Monitor has a Windows API interface through the GetPrinterData and GetPrinter functions. It also offers event notification when status of the printer changes.
All the default Windows status responses can also be scripted with WMI scripts.

Following we will call the Language Monitor just LM.

See description and programming example in Appendix C.

Windows APIs for Communication with the Printer

In order to make bi-directional communication easier and also compatible to more than one printer of the same kind on a specific PC, we implemented the LM GetPrinterData function. This is a Windows API described in the Windows documentation. To retrieve immediate printer status from the Spooler you can also use the GetPrinter function. The GetPrinterData function is preferred over GetPrinter due to the fact that with GetPrinterData, all statuses and errors display, while with GetPrinter, only printer errors display.

GetPrinterData

The GetPrinterData function retrieves configuration data for the specified printer or print server. See the Microsoft documentation (http://msdn.microsoft.com/en-us/library/dd144912(VS.85).aspx) for more information on how to use this function.

Note: See Appendix B for available key words.

Note: You can set any of the key values with new entries using the SetPrinterData function. See the Microsoft documentation (http://msdn.microsoft.com/en-us/library/dd145083(VS.85).aspx) for more information on how to use this function.

GetPrinter

The GetPrinter function retrieves information about the specified printer. See the Microsoft documentation (http://msdn.microsoft.com/en-us/library/dd144911(VS.85).aspx) for more information on how to use this function.

Note: Zebra Printer status: It is recommended to use the PRINTER_INFO_3 structure to inquire for the printer status presented by the LM.

Note: The spooler status is changed by SetPort. When using SetPort with custom messages, you can't set these to be displayed or used by the spooler. This is a known bug; "SetPort doesn't work with custom status messages." (Microsoft)

Therefore, all custom messages will be declared as PRINTER_STATUS_ERROR and a KPL value is placed in the ExternalError key. The custom messages are only accessible through the GetPrinterData function.

Event Notification

In order to eliminate the need for a program to continuously poll the printer for status, Zebra implemented an Event notification in the LM.
This notification, used together with the WaitForMultipleObjects Windows API function, enables applications to react on status changes rather than looking for status periodically.
When the internal polling thread recognizes a status change or error then it will fire an event, either an error or a status event.
The Application can open an event object to the LM events and initialize the "Wait for event" function.
The necessary event names can be extracted from the registry with the GetPrinterData API function.
When an event occurs, call the GetPrinterData function and you get the error or status condition returned.
See Appendix B for the keywords.
See the Appendix C for a programming sample.

Status update in Windows "Printers and Faxes" or "Devices and Printer"

In the case that the printer is not printing the status will be checked every 10 seconds (depending on the setting of the READ_THREAD_IDLE_SLEEP key in the LM registry setting). During printing and on error the status will be checked more frequently.

The same status that can be gathered with the GetPrinterData or GetPrinter API will be displayed in the Printer folder.

Note: In some cases it may be possible that the PnP ping is not properly executed on the system and therefore the idle thread of the LM is not activated after a power off situation of the printer. In this case the LM will be reactivated the next time a print job is executed.

Appendix A

Windows Compatible Status

These statuses will also be stored in the printer ERROR key in the registry and can be extracted with GetPrinterData.

Statuses Defined in winspool.h

Table 4 • Windows Status
Windows Status Compares to Zebra Status

PRINTER_STATUS_PAPER_JAM	Paper jam (ESC ENQ 1 = NAK 1)
PRINTER_STATUS_USER_INTERVENTION	Cutter not home (ESC ENQ 1 = NAK 2)
PRINTER_STATUS_PAPER_OUT	Out of paper (ESC ENQ 1 = NAK 3)
PRINTER_STATUS_DOOR_OPEN	Print head lifted (ESC ENQ 1 = NAK 4)
PRINTER_STATUS_PAPER_PROBLEM	Paper feed problem (ESC ENQ 1 = NAK 5)
PRINTER_STATUS_NOT_AVAILABLE	Temperature error (ESC ENQ 1 = NAK 6)
PRINTER_STATUS_ERROR	Presenter jam (ESC ENQ 1 =NAK 7), check ExternalError

PRINTER_STATUS_NOT_AVAILABLE	Retract jam (ESC ENQ 1 = NAK 8), check ExternalError
PRINTER_STATUS_NOT_AVAILABLE	Black mark not found (ESC ENQ 1 = NAK 10), check ExternalError
PRINTER_STATUS_NOT_AVAILABLE	Black mark calibration error (ESC ENQ 1 = NAK 11), check ExternalError
PRINTER_STATUS_NOT_AVAILABLE	Index error (ESC ENQ 1 = NAK 12), check ExternalError
PRINTER_STATUS_NOT_AVAILABLE	Checksum error (ESC ENQ 1 = NAK 13), check ExternalError
PRINTER_STATUS_NOT_AVAILABLE	Wrong firmware (ESC ENQ 1 = NAK 14), check ExternalError
PRINTER_STATUS_NOT_AVAILABLE	Retract occurred (ESC ENQ 1 = NAK 16), check ExternalError
PRINTER_STATUS_NOT_AVAILABLE	Paused (ESC ENQ 1 = NAK 17), check ExternalError
PRINTER_STATUS_TONER_LOW	Paper near end (ESC ENQ 6)
PRINTER_STATUS_NO_TONER	Weekend paper status (ESC ENQ 6) (only for TTP 7030 and TTP 8200 with special hardware)
PRINTER_STATUS_OUTPUT_BIN_FULL	Paper in presenter (ESC ENQ 6)

Table 5 • Status definition in Winspool.h

#define PRINTER_STATUS_ERROR	0x00000002
#define PRINTER_STATUS_PAPER_JAM	0x00000008
#define PRINTER_STATUS_PAPER_OUT	0x00000010
#define PRINTER_STATUS_PAPER_PROBLEM	0x00000040
#define PRINTER_STATUS_OFFLINE	0x00000080
#define PRINTER_STATUS_OUTPUT_BIN_FULL	0x00000800
#define PRINTER_STATUS_NOT_AVAILABLE	0x00001000
#define PRINTER_STATUS_TONER_LOW	0x00020000
#define PRINTER_STATUS_NO_TONER	0x00040000
#define PRINTER_STATUS_USER_INTERVENTION	0x00100000
#define PRINTER_STATUS_DOOR_OPEN	0x00400000

Note: In order to indicate the Kiosk printer status of Paper-near-end or Weekend-paper-status Zebra is using two for thermal Kiosk printer's unused Microsoft status codes. The codes used are PRINTER_STATUS_TONER_LOW for Paper-near-end and PRINTER_STATUS_NO_TONER for Weekend-paper-status. The Weekend-paper-status is only present with printers that have the option of two sensors on their roll holder. (See the Technical Manual for your printer for more information on the available options.)

Windows Incompatible Status

If a printer status doesn't have a corresponding Windows status the Error key will have PRINTER_STATUS_NOT_AVAILABLE set and you need to evaluate the ExternalError key.

Statuses that have a representation within the Windows status may also have an ESC ENQ 1 NAK value (see Table 4) and will be stored in the printer ExternalError key in the registry and can be extracted with GetPrinterData using the ExternalError key.
For the meanings of these NAK responses see the appropriate Technical Manual for your printer, under the ESC ENQ 1 section.

Note: Any other Windows status may be used in the future, so mask away undefined bits in your application!

Appendix B

Table 8 • GetPrinterData Key Values

Printer	DsMonitor Key Explanation	Type
DeviceID	Printer's device ID string	REG_BINARY
Error	Printer Error or Status in Windows 16-bit format	REG_DWORD
ErrorEvent	Error event name for error event trigger	REG_SZ
ExternalError	Extended status according to Appendix B	REG_DWORD
Firmware	Firmware version	REG_BINARY
PageCount	Page counter for cut pages	REG_DWORD
PCB_REV	Printers PCB revision number	REG_BINARY
PCB_SN	Printers PCB serial number	REG_BINARY
StatusEvent	Status event name for status event trigger	REG_SZ
RetractCount	Retract counter for retracted pages	REG_DWORD
DeleteJob	Flag to delete print jobs on error	REG_DWORD
Head_Temp	Head temperature (ESC ENQ B)	REG_DWORD

Appendix C

Programming example

Background

In order to incorporate the new way of status monitoring you need to get a little bit of background on what happens in a Kiosk when you print and when you should monitor your status.
Status monitoring can be handled in two different ways.
Monitor in the printing application
Monitor in a separate application

When you monitor in your printing application you would commonly look at the printer before sending a print job to see if the printer is OK and then send your print job. After the print job is signaled as being printed you would check status again to see if the printer has any errors or if the paper has been taken, etc.

Monitoring in a separate application usually doesn't allow direct interaction with the printed job so you are trying to poll the printer as often as you can to get most accurate information on what the printer is doing. This is usually a very time consuming task and you have to care for synchronizing with a current print job.

Since the latter example is most commonly used for status monitoring, we have incorporated an event notification into the LM to allow a monitoring application to do other tasks and have a separate thread listening for the printer status or error event change. When this occurs the thread is simply getting the status and reporting this back to the main program or doing any other kind of reporting.

To accommodate this notification for all error and status changes we incorporated two mechanisms in the LM.

1. Monitoring while printing

We implemented status monitoring in the internal printing structure of the LM. When you open a Document, print it and close the Document again the LM will check the printer status before and after printing and will also react to write errors if such occur. Then it will set the printer status and raise the error event.

2. Monitoring while idle

We implemented an internal status thread which polls the printer when it is idle in a predefined cycle and provides changed status information in the same manner. It will set the status and raise an error or status event. Therefore, it is not necessary to implement your own monitoring loop. You can simply wait for an event in your application's idle loop.

Implementation in calling application

1. **Step: Open the Printer**

The first step of your implementation is to open the printer you want to monitor and get the Error event and Status even name.

```
bRet = OpenPrinter(m_csPrinter.GetBuffer(1), &hPrinter, &pd);
…
```

```
if ((dRet = GetPrinterData(hPrinter, "ErrorEventName", &dType, (LPBYTE)cTmp, 100,
&dNeeded))!=ERROR_SUCCESS)
…
if ((dRet = GetPrinterData(hPrinter, "StatusEventName", &dType, (LPBYTE)cTmp, 100,
&dNeeded))!=ERROR_SUCCESS)
…
```

2. **Step: Open the Event Handles**

Then you open the two event handles and fill these handles into a structure you will pass on to the new thread.

```
typedef struct _CStatusThreadInfo
{
        HWND  myHwnd;
        DWORD         dSleepTime;
        HANDLE        hPrinter;
        HANDLE        hError;
        HANDLE        hStatus;
        BOOL    m_hStatusEventKillThread;
} CStatusThreadInfo;
…
if ((cTi.hError = OpenEvent(SYNCHRONIZE, TRUE, m_csErrorEvent))==NULL)
…
if ((cTi.hStatus = OpenEvent(SYNCHRONIZE, TRUE, m_csStatusEvent))==NULL)
```

3. **Step: Start Monitoring**

When all this is done you can start your monitoring thread.

```
m_StatusThread = AfxBeginThread( StatusThreadProc, &cTi, THREAD_PRIORITY_NORMAL,0,0,NULL);
```

Implementation in monitor thread

4. **Step: Fill Event Arrays**

In the monitoring thread you create and fill and array of handles with the error and status event handle.

```
        myHandle[0] = pInfo->hError;
        myHandle[1] = pInfo->hStatus;
```

5. **Step: Start the Waiting Loop**

Then you are ready to start the waiting loop.

```
for ( ; ; )
{
    if (pInfo->m_hStatusEventKillThread)
    {
        OutputDebugStringA("### [Thread msg.] Kill thread...\n");
        pInfo->m_hStatusEventKillThread = FALSE;
        AfxEndThread( 1 );
        return 1;
    }
    if ((dwRet = WaitForMultipleObjects(2, myHandle, FALSE, pInfo->dSleepTime))!=WAIT_FAILED)
    {
        if (dwRet==WAIT_OBJECT_0 || dwRet==WAIT_OBJECT_0+1)
        {
            if ((dwRet = GetPrinterData(hPrinter, "Error", &dType, (LPBYTE)&dwResult, sizeof(dwResult),
&dNeeded))!=ERROR_SUCCESS)
            {
                    sprintf( str, "### [Status Thread error %d] read [%08X]\n", dwRet, dwResult);
                    OutputDebugStringA(str);
            }
            sprintf( str, "### [Status Thread] read [%08X]\n", dwResult);
            OutputDebugStringA(str);
            SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0, (LPARAM)(str));
            if (dwResult & 0x00000000)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_OK"));
            if (dwResult & PRINTER_STATUS_ERROR)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_ERROR"));
            if (dwResult & PRINTER_STATUS_PENDING_DELETION)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_PENDING_DELETION"));
            if (dwResult & PRINTER_STATUS_PAPER_JAM)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_PAPER_JAM"));
            if (dwResult & PRINTER_STATUS_PAPER_OUT)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_PAPER_OUT"));
            if (dwResult & PRINTER_STATUS_PAPER_PROBLEM)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_PAPER_PROBLEM"));
            if (dwResult & PRINTER_STATUS_OFFLINE)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_OFFLINE"));
            if (dwResult & PRINTER_STATUS_IO_ACTIVE)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_IO_ACTIVE"));
            if (dwResult & PRINTER_STATUS_BUSY)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_BUSY"));
            if (dwResult & PRINTER_STATUS_PRINTING)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_PRINTING"));
            if (dwResult & PRINTER_STATUS_OUTPUT_BIN_FULL)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_OUTPUT_BIN_FULL"));
            if (dwResult & PRINTER_STATUS_PROCESSING)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_PROCESSING"));
            if (dwResult & PRINTER_STATUS_USER_INTERVENTION)
               SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
          (LPARAM)("PRINTER_STATUS_USER_INTERVENTION"));
            if (dwResult & PRINTER_STATUS_DOOR_OPEN)
```

```
            SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
        (LPARAM)("PRINTER_STATUS_DOOR_OPEN"));

          if (dwResult & PRINTER_STATUS_TONER_LOW)
            SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
        (LPARAM)("PRINTER_STATUS_PAPER_NEAR_END"));
          if (dwResult & PRINTER_STATUS_NO_TONER)
            SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
        (LPARAM)("PRINTER_STATUS_PAPER_WEEKEND"));
          if (dwResult & PRINTER_STATUS_NOT_AVAILABLE)
          {
            SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
        (LPARAM)("PRINTER_STATUS_EXTERNAL_ERROR"));
            if ((dwRet = GetPrinterData(hPrinter, "ExternalError", &dType, (LPBYTE)dwResult, sizeof(dwResult),
        &dNeeded))!=ERROR_SUCCESS)
                {
                        sprintf( str, "### [Status Thread error %d] read [%08X]\n", dwRet, dwResult);
                        OutputDebugStringA(str);
                }
                sprintf( str, "### [Status Thread External Error] read [%08X]\n", dwResult);
                OutputDebugStringA(str);
                SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
(LPARAM)(str));
            }
        }
        else
            SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
(LPARAM)("Timeout"));
    }
    else
    {
        dwRet = GetLastError();
        sprintf( str, "### Wait function failed! [%d]\n", dwRet);
        OutputDebugStringA(str);
    }
}
```

When an event occurs you need to get the status with GetPrinterData using the “Error” key and decode the result according to the sample or any way you feel necessary. In any case you can send a message or do any form of status reporting you want to do.

WMI script to get basic status

```
' VBScript source code
ttpname=""
strComputer = "."
Set objWMIService = GetObject("winmgmts:" _
   & "{impersonationLevel=impersonate}!\\" & strComputer & "\root\cimv2")
Set wbemObjectSet = objWMIService.ExecQuery("SELECT * FROM Win32_Printer")
For Each wbemObject In wbemObjectSet
   if wbemObject.Default = TRUE then
      ttpname = wbemObject.Caption
          Wscript.Echo  "Printer " & ttpname
          Select Case wbemObject.PrinterStatus
          Case 1
                  strPrinterStatus = "Other"
            strExtendedPrinterStatus  = wbemObject.ExtendedPrinterStatus
              Case 2
            strPrinterStatus = "Unknown"
              Case 3
            strPrinterStatus = "Idle"
              Case 4
```

```
            strPrinterStatus = "Printing"
               Case 5
            strPrinterStatus = "Warmup"
               Case 6
            strPrinterStatus = "Stopped printing"
               Case 7
            strPrinterStatus = "Offline"
      End Select
         Wscript.Echo  "Printer Status: " & strPrinterStatus
      Select Case wbemObject.DetectedErrorState
         Case 0
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " Unknown"
         case 1
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " Other"
         case 2
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " No Error"
         case 3
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " Low Paper"
         case 4
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " No Paper"
         case 5
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " Low Toner"
         case 6
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " No Toner"
         case 7
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " Door Open"
         case 8
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " Jammed"
         case 9
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " Offline "
         case 10
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " Service Requested"
         case 11
            Wscript.Echo  "DetectedErrorState: " & wbemObject.DetectedErrorState & " Output Bin Full"
      End Select

      Select Case wbemObject.ExtendedDetectedErrorState
         Case 0
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Unknown"
         case 1
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Other"
         case 2
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " No Error"
         case 3
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Low
Paper"
         case 4
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " No Paper"
         case 5
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Low
Toner"
         case 6
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " No Toner"
         case 7
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Door
Open"
         case 8
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Jammed"
         case 9
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Service
Requested"
         case 10
```

```
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Output
Bin Full"
         case 11
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Paper
Problem"
         case 12
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Cannot
Print Page"
         case 13
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " User
Intervantion Required"
         case 14
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Out of
Memory"
         case 15
            Wscript.Echo  "ExtendedDetectedErrorState: " & wbemObject.ExtendedDetectedErrorState & " Server
Unknown"
      End Select

   end if
Next
Wscript.Echo  "Printer " & ttpname
```