

TTP Language Monitor Implementation



Acknowledgments

Adobe and *Acrobat* are trademarks of Adobe Systems Incorporated
Windows, *Windows NT*, and *Windows XP* are trademarks of Microsoft Corporation

This is a publication of
Swecoin - A Zebra Technologies Company
30 Plan Way, Warwick, RI, 02886

Phone +1 401-848-2000
Fax +1 401-848-2058
E-mail SwecoinTechsupport@zebra.com
SwecoinSales@zebra.com
Web site <http://www.swecoinus.com>

© Swecoin 2007

All rights reserved. Reproduction in whole or in parts is prohibited without written consent of the copyright owner. We have taken great care to ensure that the information in this manual is correct and complete. However, if you discover any errors or omissions, or if you wish to make suggestions for improvements, you are welcome to send your comments to us. Swecoin US disclaims any liability resulting from the use of this information and reserves the right to make changes without notice.

Edition G, March 2007
Revision 2.3.9
Printed in USA

Table of Contents

Table of Contents.....	3
Introduction	4
Functionality.....	4
GetPrinterData.....	5
GetPrinter	7
Status response for GetPrinterData and GetPrinter	9
Windows compatible status.....	9
Swecoin extended status.....	10
Event notification.....	12
GetPrinter.....	12
GetPrinterData	12
Registry entries	13
In the Language Monitor Key	13
In the Printer Key.....	14
Programming examples.....	15
Background.....	15
Implementation	16
1. Step.....	16
2. Step.....	16
3. Step.....	16
4. Step.....	16
5. Step.....	16
Tables Overview.....	19

Introduction

The TTP Language Monitor (LM) enables an application to read status data back from the printer. The LM is a low-level driver add-on and communicates on the Application level with the Spooler and on the driver level with the Port Monitor.

In order to accomplish multi-threaded application access, the LM needed to be rewritten and is now using an internal polling thread, which in turn sends an event when the status has changed.

An ATL-based wrapper object can then inquire the current status with a proprietary function call to the LM. On the other hand, Windows applications no longer have to use our proprietary LM functions; we have implemented printer status integrated with the spooler (GetPrinter function) and another Windows API: GetPrinterData.

It is recommended to use the GetPrinterData function over the GetPrinter function since the GetPrinter function only retrieves error status and not informational statuses from the printer.

Functionality

The LM provides two functionalities.

The implementation of the GetPrinterData function

This is a Windows API which retrieves registry information about the printer. The LM takes care of placing all active status changes into the respective registry keys (described later) and the application can access these through this function.

A driver event notification when the status changes

The LM will signal a non-error status change one time, even though the changed status will persist. The LM will signal an error as long as the error persists. The status can be inquired with either of the above functions.

In the Windows Printing and Spooler Function reference you will find also another function GetPrinter which provides actual error status from the printer as well but does not provide non-error (informational) status like paper low or near-end. It is therefore recommended to always use get status with GetPrinterData.

Note:

The GetJob function is not recommended to be used since a print job causing an error in the printer will be immediately deleted by the LM and can't be used for further status inquiries.

GetPrinterData

The **GetPrinterData** function retrieves configuration data for the specified printer or print server. The configuration and status data are placed into the language monitor-specific registry keys by the LM.

Windows 2000/XP: Calling **GetPrinterData** is equivalent to calling the [GetPrinterDataEx](#) function with the *pKeyName* parameter set to "PrinterDriverData".

```
DWORD GetPrinterData(  
    HANDLE hPrinter,    // handle to printer or print server  
    LPTSTR pValueName, // value name  
    LPDWORD pType,     // data type  
    LPBYTE pData,     // configuration data buffer  
    DWORD nSize,      // size of configuration data buffer  
    LPDWORD pcbNeeded // bytes received or required  
);
```

Parameters

hPrinter

[in] Handle to the printer or print server for which the function retrieves configuration data. Use the [OpenPrinter](#) or [AddPrinter](#) function to retrieve a printer handle.

pValueName

[in] Pointer to a null-terminated string that identifies the data to retrieve.

For printers, this string is the name of a registry value under the printer's "PrinterDriverData" key in the registry.

For print servers, this string is one of the predefined strings listed in the following Remarks section.

pType

[out] Pointer to a variable that receives the type of data retrieved. The function returns the type specified in the [SetPrinterData](#) or [SetPrinterDataEx](#) call when the data was stored. This parameter can be NULL if you don't need the information.

pData

[out] Pointer to a buffer that receives the configuration data.

nSize

[in] Specifies the size, in bytes, of the buffer pointed to by *pData*.

pcbNeeded

[out] Pointer to a variable that receives the size, in bytes, of the configuration data. If the buffer size specified by *nSize* is too small, the function returns ERROR_MORE_DATA, and *pcbNeeded* indicates the required buffer size.

Return Values

If the function succeeds, the return value is ERROR_SUCCESS. If the function fails, the return value is an error value.

Remarks

Swecoin Printer status: All the Registry entries listed below can be inquired with the GetPrinterData function. The LM will be providing the actual printer information within these registry keys.

Printer DsMonitor Key	Explanation	Type
DeviceID	Printer's device ID string	REG_BINARY
ERROR	Printer Error or Status in Windows 16-bit format (see Table 3)	REG_DWORD
ErrorEvent	Error event name for error event trigger	REG_SZ
EXTERNALERROR	Extended status according to Table 5	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
Monitor Key	Explanation	Type
ACK_SLEEP	ACK marker sleep time	REG_DWORD
READ_SLEEP	Sleep time before a read after write	REG_DWORD
READ_THREAD_SLEEP	Read thread sleep time	REG_DWORD
READ_REPEAT	Read repeat count.	REG_DWORD

* Note: This is not the same as the printer serial number. PCB_SN is the serial number of the control board and is not related to the actual printer serial number. There is no way to obtain the printer serial number via software.

Table 1 GetPrinterData Key values

GetPrinter

The **GetPrinter** function retrieves information about a specified printer.

```
BOOL GetPrinter(  
    HANDLE hPrinter,    // handle to printer  
    DWORD Level,        // information level  
    LPBYTE *pPrinter,   // printer information buffer  
    DWORD cbBuf,        // size of buffer  
    LPDWORD pcbNeeded   // bytes received or required  
);
```

Parameters

hPrinter

[in] Handle to the printer for which the function retrieves information. Use the [OpenPrinter](#) or [AddPrinter](#) function to retrieve a printer handle.

Level

[in] Specifies the level or type of structure that the function stores into the buffer pointed to by *pPrinter*.

Windows 2000/XP: This value can be 1, 2, 3, 4, 5, 6, 7, 8 or 9.

pPrinter

[out] Pointer to a buffer that receives a structure containing information about the specified printer. The buffer must be large enough to receive the structure and any strings or other data to which the structure members point. If the buffer is too small, the *pcbNeeded* parameter returns the required buffer size.

The type of structure is determined by the value of *Level*.

Level	Structure
1	A PRINTER_INFO_1 structure containing general printer information.
2	A PRINTER_INFO_2 structure containing detailed information about the printer.
3	Windows NT/2000/XP: A PRINTER_INFO_3 structure containing the printer's security information.
4	Windows NT/2000/XP: A PRINTER_INFO_4 structure containing minimal printer information, including the name of the printer, the

	name of the server, and whether the printer is remote or local.
5	A PRINTER_INFO_5 structure containing printer information such as printer attributes and time-out settings.
6	Windows 2000/XP: A PRINTER_INFO_6 structure specifying the status value of a printer.
7	Windows 2000/XP: A PRINTER_INFO_7 structure that indicates whether the printer is published in the directory service.
8	Windows 2000/XP: A PRINTER_INFO_8 structure specifying the global default printer settings.
9	Windows 2000/XP: A PRINTER_INFO_9 structure specifying the per-user default printer settings.

cbBuf

[in] Specifies the size, in bytes, of the buffer pointed to by *pPrinter*.

pcbNeeded

[out] Pointer to a variable that the function sets to the size, in bytes, of the printer information. If *cbBuf* is smaller than this value, **GetPrinter** fails, and the value represents the required buffer size. If *cbBuf* is equal to or greater than this value, **GetPrinter** succeeds, and the value represents the number of bytes stored in the buffer.


Return Values

If the function succeeds, the return value is a nonzero value.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

Remarks

Swecoin Printer status: It is recommended to use the [PRINTER_INFO_2](#) structure to inquire for the printer error status presented by the LM.

 **Security Alert** The **pDevMode** member in the **PRINTER_INFO_2**, **PRINTER_INFO_8**, and **PRINTER_INFO_9** structures can be NULL. When this happens, the printer is unusable until the driver is reinstalled successfully.

Windows NT/2000/XP: For the [PRINTER_INFO_2](#) and [PRINTER_INFO_3](#) structures that contain a pointer to a security descriptor, the function retrieves only

those components of the security descriptor that the caller has permission to read. To retrieve particular security descriptor components, you must specify the necessary access rights when you call the [OpenPrinter](#) function to retrieve a handle to the printer. The following table shows the access rights required to read the various security descriptor components.

Access Right	Security Descriptor Component
READ_CONTROL	Owner Primary group Discretionary access-control list (DACL)
ACCESS_SYSTEM_SECURITY	System access-control list (SACL)

Status response for GetPrinterData and GetPrinter

Windows compatible status

With Auto Status or in the ERROR registry key.
Used with GetPrinterData and GetPrinter.

Windows status	compares to Swecoin status
PRINTER_STATUS_PAPER_JAM	Paper jam (ESC ENQ 1 = 1)
PRINTER_STATUS_PAPER_PROBLEM	Paper feed problem (ESC ENQ 1 = 5)
PRINTER_STATUS_DOOR_OPEN	Print head lifted (ESC ENQ 1 = 4)
PRINTER_STATUS_PAPER_OUT	Out of paper (ESC ENQ 1 = 3)
PRINTER_STATUS_USER_INTERVENTION	Cutter not home (ESC ENQ 1 = 2)
PRINTER_STATUS_PAPER_NEAR_END	Paper near end (ESC ENQ 6)
PRINTER_STATUS_PAPER_WEEKEND	Weekend paper status (ESC ENQ 6)
PRINTER_STATUS_ERROR	Temperature error (ESC ENQ 1 = 6) or an Undefined error

Table 2: Windows vs. Swecoin status

Note:

Below statuses are defined 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_USER_INTERVENTION	0x00100000
#define PRINTER_STATUS_DOOR_OPEN	0x00400000

Table 3: Winspool statuses

Below statuses are defined by Swecoin.

#define PRINTER_STATUS_PAPER_NEAR_END	0x02000000
#define PRINTER_STATUS_PAPER_WEEKEND	0x04000000
#define PRINTER_STATUS_EXTERNAL_ERROR	0x10000000

Table 4: Swecoin statuses

Note:

The status PRINTER_STATUS_PAPER_AT_PRESENTER was replaced in the Language Monitor version later then 1.5.10.42 with the Winspool status PRINTER_STATUS_OUTPUT_BIN_FULL.

Note:

If you are getting one of the following statuses you should check the Ext Auto Status.

PRINTER_STATUS_ERROR
PRINTER_STATUS_USER_INTERVENTION
PRINTER_STATUS_EXTERNAL_ERROR

Swecoin extended status

With Ext Auto Status or in the EXTERNALERROR registry key.
Used only with GetPrinterData.

Below statuses are External Error statuses defined by Swecoin.

#define NAK6	0x00000001
#define NAK7	0x00000002
#define NAK12	0x00000004
#define NAK13	0x00000008
#define NAK14	0x00000010
#define NAK16	0x00000020
#define BUFFEROVERFLOW	0x00000040
#define NAK17	0x00000080
#define NAK8	0x00000100

Table 5: External error statuses

Note:
Any other Windows status can be used in the future!

Event notification

When the internal polling thread recognizes a status change or error then it will fire an event, either an error or a status event.

There are two ways to get the status at this time.

GetPrinter

When an event occurs, you can collect the error status by calling this Windows API. This function is limited in use to error-only status, since it is not possible to retrieve printer low paper and near-end paper status; these are informational only and not real errors.

GetPrinterData

When an event occurs, you can collect any status by calling this Windows API with ERROR or EXTERNALERROR key selected.

Note:

The Windows function GetJob does not work with our Language Monitor since the job creating the error will be deleted by the LM. The print job failing will always be lost whether partly printed or not printed at all!

Note:

To extract registry information you need to call the GetPrinterData function with the appropriate registry key, e.g. ErrorEvent or StatusEvent as the key value to extract the event name to open the event handle.

Registry entries

In the Language Monitor Key

ACK_SLEEP = REG_DWORD	00000064	(100 decimal)
DeleteJob = REG_DWORD	00000001	(1 decimal)
Driver = REG_SZ	"10x0MON.DLL" or the equivalent LM for the specific printer	
READ_REPEAT = REG_DWORD	00000001	(1 decimal)
READ_SLEEP = REG_DWORD	00000064	(100 decimal)
READ_THREAD_SLEEP = REG_DWORD	000005dc	(1500 decimal)

Table 6: Registry entries in the Language Monitor Key

For older drivers (deployed in 2004) the following two entries are still available.

ErrorEvent = REG_SZ "ErrorEvent1" or the equivalent name for the specific printer

StatusEvent = REG_SZ "StatusEvent1" or the equivalent name for the specific printer

The READ_THREAD_SLEEP controls the auto status inquiry time.

When the LM initializes, it starts a read thread which, in turn, runs until the LM closes down.

During print idle time, the read thread is inquiring the status from the printer and signals a status change to the listening application which has implemented a WaitForMultipleObjects function.

ACK_SLEEP controls the sleep time in case of a page hold inquiry.

The page hold function is only available for the TTP8x00 and TTP20x0 series of printers and is used together with a driver setting in the 8x00 / 20x0 series driver.

READ_SLEEP controls the sleep time between a status inquiry and the successive read call.

When the LM is inquiring for status the printer needs time to gather the needed information and therefore can't react immediately. Since some printers react faster than others and the speed of the PC system and the used OS are also variables which differ, this key was implemented to fine-tune the application. You may wish to increase this value if you find that your status calls are returning ERROR or NULL.

READ_REPEAT controls the amount of re-tries on a failed read inside the LM read function.

DeleteJob controls what the LM does with pending print jobs when an error appears. In the case this value is 1 the LM will delete all pending print jobs and if it is 0 it will hold the pending jobs and stop any spooler activities until the error is resolved.

In the Printer Key

In newer drivers (after September 2004), the Error and Status event names have been moved into a separate key (DsMonitor) in the Printers section. This is to allow multiple printers of the same kind to reflect status and error conditions.

DeviceID = REG_SZ	the printer's DeviceID in hex value
ERROR = REG_BINARY	the printer's error status according to Windows status values
EXTERNALERROR = REG_BINARY	the printers error status according to ESC ENQ 1 (Swecoin-specific)
ErrorEvent = REG_SZ	"ErrorEvent1" or the equivalent name for the specific printer
PAGECOUNT = REG_DWORD	current page count (this is the number of pages printed with this particular installation of driver, not necessarily the number of pages printed on a given printer.
PCB_REV = REG_BINARY	printer's PCB revision
PCB_SN = REG_BINARY	printer's PCB serial number (not the serial number which is located on a label on the side of the printer)
StatusEvent = REG_SZ	"StatusEvent1" or the equivalent name for the specific printer

Table : Registry entries in the Printer Key

ErrorEvent and StatusEvent are the event names used in the LM to signal the current status changes. When you open an event with the listed name you can wait for this event to happen and then inquire with the "Auto Status" or "Ext Auto Status" for the status value.

Note:

The difference between "Auto Status" and "Ext Auto Status" is the return value. In the case of "Auto Status" you will get a DWORD back that is also stored in the ERROR value in the registry and in the case of "Ext Auto Status" you will get two DWORD's back where the first is the ERROR and the second the EXTERNALERROR value in the registry which reflects the printers ESC ENQ 1 value (see the appropriate Technical Manual).

Note:

When changes are made in the registry it is necessary to reboot the PC in order for the changed parameters to take effect.

Programming examples

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 such things as not to interfere with a current print job.

Since the latter example is most commonly used for status monitoring, we have incorporated the even notification into the Language Monitor 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 Language Monitor. 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

The sample program used to demonstrate the status monitoring with GetPrinterData is Monitor2 and the main dialog “Monitor2Dlg.cpp” implements the necessary steps.

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. You can find this in the BOOL CMonitor2Dlg::OnInitDialog() function.

```
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. You can find this in the BOOL CMonitor2Dlg::OnInitDialog() function.

```
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. You can find this in the void CMonitor2Dlg::OnAutoStatus() function.

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

4. Step: Fill Event Arrays

In the monitoring thread you create and fill an array of handles with the error and status event handle. You can find this in the UINT StatusThreadProc(LPVOID pParam) function.

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

5. Step: Start the Waiting Loop

Then you are ready to start the waiting loop which you find in the UINT StatusThreadProc(LPVOID pParam) function.

```

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_PAPER_NEAR_END)
                SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
                (LPARAM)("PRINTER_STATUS_PAPER_NEAR_END"));
        }
    }
}

```

```

if (dwResult & PRINTER_STATUS_PAPER_WEEKEND)
    SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
        (LPARAM)("PRINTER_STATUS_PAPER_WEEKEND"));
if (dwResult & PRINTER_STATUS_PAPER_PRESENTER)
    SendMessage(GetDlgItem((HWND)pInfo->myHwnd, IDC_Status), WM_SETTEXT, 0,
        (LPARAM)("PRINTER_STATUS_PAPER_PRESENTER"));
if (dwResult & PRINTER_STATUS_EXTERNAL_ERROR)
    {
    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` 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.

Tables Overview

Table 2 GetPrinterData Key values	6
Table 3: Windows vs. Swecoin status.....	9
Table 4: Winspool statuses.....	10
Table 5: Swecoin statuses	10
Table 6: External error statuses	10
Table 8: Registry entries in the Language Monitor Key.....	13