基于HOOK 和 状态轮询的 打印机监控，内容抓取

原创

h2052519 于 2018-03-17 16:45:26 发布
分类专栏： C 文章标签： C

基于HOOK 和 状态轮询的 打印机监控，内容抓取，网上例子要么都是介绍如何挂钩子监控，要么是轮询查询打印机状态，很少有两者联合起来监控，兼抓取内容的。先贴下代码。

```
// PrinterHook.cpp : 定义 DLL 应用程序的导出函数。
//

#include "stdafx.h"
#include "PrinterHook.h"

#define PRINTER_PIPE_NAME  "\\\\.\\Pipe\\PrinterPipeName"
//老的函数指针声明
typedef HDC  (WINAPI* pOldCreateDCA)(LPCSTR pwszDriver, LPCSTR pwszDevice, LPCSTR pszPort,  CONST DEVMODEA
typedef HDC  (WINAPI* pOldCreateDCW)(LPCWSTR pwszDriver, LPCWSTR pwszDevice, LPCWSTR pszPort,  CONST DEVMOD
typedef int  (WINAPI* pOldStartDocA)( HDC hdc,  CONST DOCINFOA *lpdi);
typedef int  (WINAPI* pOldStartDocW)( HDC hdc,  CONST DOCINFOW *lpdi);
typedef int (WINAPI* pOldEndDoc)(HDC hdc);
typedef int  (WINAPI* pOldStartPage)(HDC hdc);
typedef int  (WINAPI* pOldEndtPage)(HDC hdc);
//文字绘制HOOK
typedef BOOL (WINAPI* pOldTextOutA)( HDC hdc, int x, int y,  LPCSTR lpString, int c);
typedef BOOL (WINAPI* pOldTextOutW)( HDC hdc, int x, int y,  LPCWSTR lpString, int c);
typedef int  (WINAPI* pOldDrawTextExA)(HDC hdc,LPSTR lpchText,int cchText,LPRECT lprc, UINT format,LPDRAWTE
typedef int  (WINAPI* pOldDrawTextExW)(HDC hdc,LPWSTR lpchText,int cchText,LPRECT lprc, UINT format,LPDRAWT
typedef BOOL (WINAPI* pOldExtTextOutA)(  HDC hdc,  int x,  int y,  UINT options,  CONST RECT * lprect,  LPC
typedef BOOL (WINAPI* pOldExtTextOutW)(  HDC hdc,  int x,  int y,  UINT options,  CONST RECT * lprect,  LPC
typedef BOOL (WINAPI* pOldPolyTextOutA)( HDC hdc, CONST POLYTEXTA * ppt,  int nstrings);
typedef BOOL (WINAPI* pOldPolyTextOutW)( HDC hdc, CONST POLYTEXTW * ppt,  int nstrings);
typedef int  (WINAPI* pOldDrawTextA)(HDC hdc, LPCSTR lpchText,int cchText, LPRECT lprc,UINT format);
typedef int  (WINAPI* pOldDrawTextW)(HDC hdc, LPCWSTR lpchText,int cchText, LPRECT lprc,UINT format);
typedef LONG (WINAPI* pOldTabbedTextOutA)(HDC hdc,int x,int y,LPCSTR lpString,int chCount,int nTabPositions
typedef LONG (WINAPI* pOldTabbedTextOutW)(HDC hdc,int x,int y,LPCWSTR lpString,int chCount,int nTabPosition

typedef struct _PrintFileInfo
{
    FILE*       pFile;
    DWORD       dwPage;
    std::string strFileTime;
    std::string strFileName;
    std::string strDocName;
}PrintFileInfo;
```

```
enum
{
    emStartCheck,
    emReturnResult
};

PVOID g_pOldCreateDCA = NULL;
PVOID g_pOldCreateDCW = NULL;
PVOID g_pOldStartDocA = NULL;
PVOID g_pOldStartDocW = NULL;
PVOID g_pOldEndDoc = NULL;
PVOID g_pOldStartPage = NULL;
PVOID g_pOldEndPage = NULL;
PVOID g_pOldTextOutA = NULL;
PVOID g_pOldTextOutW = NULL;
PVOID g_pOldDrawTextExA = NULL;
PVOID g_pOldDrawTextExW = NULL;
PVOID g_pOldExtTextOutA = NULL;
PVOID g_pOldExtTextOutW = NULL;
PVOID g_pOldPolyTextOutA = NULL;
PVOID g_pOldPolyTextOutW = NULL;
PVOID g_pOldDrawTextA = NULL;
PVOID g_pOldDrawTextW = NULL;
PVOID g_pOldTabbedTextOutA = NULL;
PVOID g_pOldTabbedTextOutW = NULL;

PrintFileInfo PrinterFileInfo = {NULL};
HDC hDc = NULL;

HDC  WINAPI CreateDCAHook(  LPCSTR pwszDriver,  LPCSTR pwszDevice,  LPCSTR pszPort,  CONST DEVMODEA * pdm);
HDC  WINAPI CreateDCWHook(  LPCWSTR pwszDriver,  LPCWSTR pwszDevice,  LPCWSTR pszPort,  CONST DEVMODEW * pd

int WINAPI StartPageHook(HDC hdc);
int WINAPI EndPageHook(HDC hdc);

int WINAPI StartDocAHook( HDC hdc,  CONST DOCINFOA *lpdi);
int WINAPI StartDocWHook( HDC hdc,  CONST DOCINFOW *lpdi);

int WINAPI  EndDocHook(HDC hdc);

BOOL  WINAPI TextOutAHook( HDC hdc, int x, int y,  LPCSTR lpString, int c);
BOOL  WINAPI TextOutWHook( HDC hdc, int x, int y,  LPCWSTR lpString, int c);

int  WINAPI DrawTextExAHook( HDC hdc, LPSTR lpchText,int cchText,LPRECT lprc,UINT format,LPDRAWTEXTPARAMS l
int  WINAPI DrawTextExWHook( HDC hdc, LPWSTR lpchText,int cchText,LPRECT lprc,UINT format,LPDRAWTEXTPARAMS

BOOL WINAPI ExtTextOutAHook(  HDC hdc,  int x,  int y,  UINT options,  CONST RECT * lprect,  LPCSTR lpStrin
BOOL WINAPI ExtTextOutWHook(  HDC hdc,  int x,  int y,  UINT options,  CONST RECT * lprect,  LPCWSTR lpStri

BOOL WINAPI PolyTextOutAHook( HDC hdc, CONST POLYTEXTA * ppt,  int nstrings);
BOOL WINAPI PolyTextOutWHook( HDC hdc, CONST POLYTEXTW * ppt,  int nstrings);
int  WINAPI DrawTextAHook(HDC hdc, LPCSTR lpchText,int cchText, LPRECT lprc,UINT format);
int  WINAPI DrawTextWHook(HDC hdc, LPCWSTR lpchText,int cchText, LPRECT lprc,UINT format);
LONG WINAPI TabbedTextOutAHook(HDC hdc,int x,int y,LPCSTR lpString,int chCount,int nTabPositions,CONST INT
LONG WINAPI TabbedTextOutWHook(HDC hdc,int x,int y,LPCWSTR lpString,int chCount,int nTabPositions,CONST INT

 PrinterHook* PrinterHook::m_pInstance = NULL;

 PrinterHook::CGarbo PrinterHook::Garbo;
```

```
PrinterHook::PrinterHook()
{

}
PrinterHook::~PrinterHook()
{

}

string  U2A(const wchar_t* szU)
{
    int nRetCode = (int)WideCharToMultiByte (CP_ACP, 0, szU, -1, 0, 0, NULL, NULL) ;
    if ( nRetCode ==0 )
    {
        return "";
    }
    std::string str(nRetCode-1, '\0');
    WideCharToMultiByte (CP_ACP, 0, szU, -1, (char*)(str.c_str()), nRetCode, NULL, NULL) ;
    return str;
}

void WriteContextTextA(LPCSTR lpString)
{
    if (PrinterFileInfo.pFile != NULL)
    {
        fwrite(lpString,sizeof(char),strlen(lpString),PrinterFileInfo.pFile);
    }
}

string GetDocName(std::string& strDocName)
{
    std::string strName = "";
    if (!strDocName.empty())
    {
        strName = strDocName;
        size_t pos = strName.find_last_of('\\');
        strName = strName.substr(pos+1,strName.length());
        pos = strName.find_last_of('.');
        if (pos != std::string::npos)
        {
            strName = strName.substr(0,pos);
        }
    }
    return strName;
}

void WriteContextTextW(LPCWSTR lpString)
{
    if (PrinterFileInfo.pFile != NULL)
    {
        std::string str = U2A(lpString);
        fwrite(str.c_str(),sizeof(char),str.length(),PrinterFileInfo.pFile);
    }
}

std::wstring GetWString(LPCWSTR lpString,int c)
{
    if (lpString == NULL || c == 0)
    {
```

```
    {
        return L"";
    }
    std::wstring strTmp = L"";
    WCHAR* wbuff = new WCHAR[c + 1];
    ZeroMemory(wbuff,sizeof(WCHAR)*c + 1);
    memcpy_s(wbuff,sizeof(WCHAR)*c,lpString,sizeof(WCHAR)*c);
    *(wbuff+c) = 0;
    strTmp = wbuff;
    strTmp.erase(0,strTmp.find_first_not_of(L" "));
    strTmp.erase(strTmp.find_last_not_of(L" ") + 1);
    delete [] wbuff;
    return strTmp;
}

std::string GetString(LPCSTR lpString,int c)
{
    if (lpString == NULL || c == 0)
    {
        return "";
    }
    std::string strTmp = "";
    CHAR* buff = new CHAR[c + 1];
    ZeroMemory(buff,sizeof(CHAR)*c + 1);
    memcpy_s(buff,sizeof(CHAR)*c,lpString,sizeof(CHAR)*c);
    *(buff+c) = 0;
    strTmp = buff;
    strTmp.erase(0,strTmp.find_first_not_of(" "));
    strTmp.erase(strTmp.find_last_not_of(" ") + 1);
    delete [] buff;
    return strTmp;
}

void PrinterHook::StartHook()
{
#define STR(str) #str
#define DETOURATTACH(dll,name) \
    {g_pOld##name = DetourFindFunction(dll,STR(name));\
     DetourAttach(&g_pOld##name,name##Hook);}\

    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());//得到当前线程
    DETOURATTACH("Gdi32.dll",CreateDCA);
    DETOURATTACH("Gdi32.dll",CreateDCW);
    DETOURATTACH("Gdi32.dll",StartDocW);
    DETOURATTACH("Gdi32.dll",StartDocA);
    DETOURATTACH("Gdi32.dll",StartPage);
    DETOURATTACH("Gdi32.dll",EndPage);
    DETOURATTACH("Gdi32.dll",EndDoc);
    //文字相关
    DETOURATTACH("Gdi32.dll",TextOutA);
    DETOURATTACH("Gdi32.dll",TextOutW);
    DETOURATTACH("User32.dll",DrawTextExA);
    DETOURATTACH("User32.dll",DrawTextExW);
    DETOURATTACH("Gdi32.dll",ExtTextOutA);
    DETOURATTACH("Gdi32.dll",ExtTextOutW);
    DETOURATTACH("Gdi32.dll",PolyTextOutA);
    DETOURATTACH("Gdi32.dll",PolyTextOutW);
    DETOURATTACH("User32.dll",DrawTextA);
    DETOURATTACH("User32.dll",DrawTextW);
```

```
    DETOURATTACH("User32.dll",TabbedTextOutA);
    DETOURATTACH("User32.dll",TabbedTextOutW);
    LONG ret=DetourTransactionCommit();
}

void PrinterHook::StopHook()
{
#define DETOURDETACH(name) \
    if (g_pOld##name) \
    {\
        DetourDetach(&g_pOld##name,name##Hook);\
    }\

    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());
    DETOURDETACH(CreateDCA);
    DETOURDETACH(CreateDCW);
    DETOURDETACH(StartPage);
    DETOURDETACH(EndPage);
    DETOURDETACH(EndDoc);
    DETOURDETACH(TextOutA);
    DETOURDETACH(TextOutW);
    DETOURDETACH(DrawTextExA);
    DETOURDETACH(DrawTextExW);
    DETOURDETACH(ExtTextOutA);
    DETOURDETACH(ExtTextOutW);
    DETOURDETACH(StartDocW);
    DETOURDETACH(StartDocA);

    DETOURDETACH(PolyTextOutA);
    DETOURDETACH(PolyTextOutW);
    DETOURDETACH(DrawTextA);
    DETOURDETACH(DrawTextW);
    DETOURDETACH(TabbedTextOutA);
    DETOURDETACH(TabbedTextOutW);
    LONG ret=DetourTransactionCommit();
}

int  PipeMsgSend(int type, BYTE* pData, int nDataLen)
{
    if (pData == NULL || nDataLen <= 0)
    {
        return -1;
    }
    CIPCClient ipcClient;
    if (!ipcClient.Connect(PRINTER_PIPE_NAME))
    {
        return -1;
    }

    ipcClient.SetUseAsyn(TRUE);
    ipcClient.SetTimeOut(1000);

    if (!ipcClient.SendIPCMessage(type, pData, nDataLen))
    {
        OutputDbg("Fun: %s, Line: %d, 发送管道数据错误.", __FUNCTION__, __LINE__);
    }
    ipcClient.DisConnect();
    return 0;
}
```

```
}

void NotifyCheckPrinter(BYTE* pByte,int nDataLen)
{
    PipeMsgSend(emStartCheck,pByte,nDataLen);
}

HDC  WINAPI CreateDCAHook(  LPCSTR pwszDriver,  LPCSTR pwszDevice,  LPCSTR pszPort,  CONST DEVMODEA * pdm)
{
 if (pdm != NULL)
 {
  int nDataLen = sizeof(DEVMODEA);
  BYTE* pByte = new BYTE[nDataLen];
  ZeroMemory(pByte, nDataLen);
  memcpy_s(pByte, nDataLen, pdm, nDataLen);
  NotifyCheckPrinter(pByte, nDataLen);
  delete[] pByte;
 }
    return ((pOldCreateDCA)g_pOldCreateDCA)(pwszDriver,pwszDevice,pszPort,pdm);
}

HDC  WINAPI CreateDCWHook(  LPCWSTR pwszDriver,  LPCWSTR pwszDevice,  LPCWSTR pszPort,  CONST DEVMODEW * pd
{
 if (pdm != NULL)
 {
  int nDataLen = sizeof(DEVMODEW);
  BYTE* pByte = new BYTE[nDataLen];
  ZeroMemory(pByte, nDataLen);
  memcpy_s(pByte, nDataLen, pdm, nDataLen);
  NotifyCheckPrinter(pByte, nDataLen);
  delete[] pByte;
 }
    return ((pOldCreateDCW)g_pOldCreateDCW)(pwszDriver,pwszDevice,pszPort,pdm);
}

int WINAPI EndPageHook(HDC hdc)
{
    hDc = NULL;
    WriteContextTextA("\r\n\r\n");
    return ((pOldEndtPage)g_pOldEndPage)(hdc);
}

int WINAPI StartPageHook(HDC hdc)
{
    hDc = hdc;
    PrinterFileInfo.dwPage++;
    return ((pOldStartPage)g_pOldStartPage)(hdc);
}

void CreateTextFile(std::string strDocName)
{
    CHAR chName[MAX_PATH] = {NULL};
    CHAR chPath[MAX_PATH] = {NULL};
    ::GetTempPath(MAX_PATH,chPath);
    if (!strDocName.empty())
    {
        std::string strName = GetDocName(strDocName);
        sprintf_s(chName,MAX_PATH,"%s%s.txt",chPath,strName.c_str());
    }
    if (PrinterFileInfo.pFile != NULL)
```

```
    {
        fclose(PrinterFileInfo.pFile);
    }
    if(chName != NULL)
    {
        PrinterFileInfo.strFileName = chName;
        errno_t err = fopen_s(&PrinterFileInfo.pFile,chName,"w");
  if (err !=0)
  {
   OutputDbg("%d\r\n", err);
  }
        OutputDbg("filename:%s\r\n",chName);
    }
}

//
//**********************************
// Method:    StartDocAHook
// FullName:  StartDocAHook
// Access:    public
// Returns:   int WINAPI
// Qualifier:
// Parameter: HDC hdc
// Parameter: CONST DOCINFOA * lpdi
//**********************************

int WINAPI StartDocAHook( HDC hdc,  CONST DOCINFOA *lpdi)
{
    PrinterFileInfo.strDocName = lpdi->lpszDocName;
    CreateTextFile(PrinterFileInfo.strDocName);
    return ((pOldStartDocA)g_pOldStartDocA)(hdc,lpdi);
}

int WINAPI StartDocWHook( HDC hdc,  CONST DOCINFOW *lpdi)
{
    PrinterFileInfo.strDocName = U2A(lpdi->lpszDocName);
    CreateTextFile(PrinterFileInfo.strDocName);
    return ((pOldStartDocW)g_pOldStartDocW)(hdc,lpdi);
}

int WINAPI EndDocHook(HDC hdc)
{
    if (PrinterFileInfo.pFile)
    {
        fclose(PrinterFileInfo.pFile);
        PrinterFileInfo.pFile = NULL;
    }
    if (!PrinterFileInfo.strFileName.empty() && !PrinterFileInfo.strDocName.empty())
    {
        SYSTEMTIME tm;
        GetLocalTime(&tm);
        CHAR szBeginTime[MAX_PATH] = {NULL};
        sprintf_s(szBeginTime,MAX_PATH,"%04d_%02d_%02d %02d_%02d_%02d",
            tm.wYear,
            tm.wMonth,
            tm.wDay,
            tm.wHour,
            tm.wMinute,
            tm.wSecond);
        PrinterFileInfo.strFileTime = szBeginTime;
```

```
        PrinterFileInfo.strFileTime = szBeginTime;
        std::string strSend = PrinterFileInfo.strFileTime;
        strSend.append(";");
        strSend.append(PrinterFileInfo.strDocName);
        strSend.append(";");
        strSend.append(PrinterFileInfo.strFileName);
  strSend.append(";");
  char buff[8] = { NULL };
  itoa(PrinterFileInfo.dwPage, buff, 10);
  strSend.append(buff);
        PipeMsgSend(emReturnResult,(BYTE*)(strSend.c_str()),strSend.length());
    }
    memset(&PrinterFileInfo,0x00,sizeof(PrinterFileInfo));
    return ((pOldEndDoc)g_pOldEndDoc)(hdc);
}

BOOL  WINAPI TextOutAHook( HDC hdc, int x, int y,  LPCSTR lpString, int c)
{
    if (hDc == hdc && lpString && c > 0)
    {
        static int sx = 0,sy = 0,suc = 0;
        static std::string strTmp = "";
        std::string strData = GetString(lpString,c);
        if (!strData.empty() && (sx != x || sy != y || suc != c || strData.compare(strTmp) != 0))
        {
            sx = x;sy = y;suc = c;strTmp = strData;
            WriteContextTextA(strData.c_str());
        }
    }
    return ((pOldTextOutA)g_pOldTextOutA)(hdc,x,y,lpString,c);
}

BOOL  WINAPI TextOutWHook( HDC hdc, int x, int y,  LPCWSTR lpString, int c)
{
    if (hDc == hdc && lpString && c > 0)
    {
        static int sx = 0,sy = 0,suc = 0;
        static std::wstring strTmp = L"";
        std::wstring strData = GetWString(lpString,c);
        if (!strData.empty() && (sx != x || sy != y || suc != c || strData.compare(strTmp) != 0))
        {
            sx = x;sy = y;suc = c;strTmp = strData;
            WriteContextTextW(strData.c_str());
        }
    }
    return ((pOldTextOutW)g_pOldTextOutW)(hdc,x,y,lpString,c);
}

int  WINAPI DrawTextExAHook( HDC hdc, LPSTR lpchText,int cchText,LPRECT lprc,UINT format,LPDRAWTEXTPARAMS l
{
    if (hDc == hdc && cchText > 0 && lpchText)
    {
        static int suc = 0;
        static std::string strTmp = "";
        std::string strData = GetString(lpchText,cchText);
        if (!strData.empty() && (suc != cchText || strData.compare(strTmp) != 0))
        {
            suc = cchText;strTmp = strData;
            WriteContextTextA(strData.c_str());
        }
```

```
    }
    return ((pOldDrawTextExA)g_pOldDrawTextExA)(hdc,lpchText,cchText,lprc,format,lpdtp);
}

int WINAPI DrawTextExWHook( HDC hdc,LPWSTR lpchText,int cchText,LPRECT lprc,UINT format,LPDRAWTEXTPARAMS lp
{
    if (hDc == hdc && cchText > 0 && lpchText)
    {
        static int suc = 0;
        static std::wstring strTmp = L"";
        std::wstring strData = GetWString(lpchText,cchText);
        if (!strData.empty() && (suc != cchText || strData.compare(strTmp) != 0))
        {
            suc = cchText;strTmp = strData;
            WriteContextTextW(strData.c_str());
        }
    }
    return ((pOldDrawTextExW)g_pOldDrawTextExW)(hdc,lpchText,cchText,lprc,format,lpdtp);
}

BOOL WINAPI ExtTextOutAHook(  HDC hdc,  int x,  int y,  UINT options,  CONST RECT * lprect,  LPCSTR lpStrin
{
    if (hDc == hdc && lpString && c > 0 && options < (ETO_PDY<<1) )
    {
        static int sx = 0,sy = 0,suc = 0;
        static std::string strTmp = "";
        std::string strData = GetString(lpString,c);
        if (!strData.empty() && (sx != x || sy != y || suc != c || strData.compare(strTmp) != 0))
        {
            sx = x;sy = y;suc = c;strTmp = strData;
            WriteContextTextA(strData.c_str());
        }
    }
    return ((pOldExtTextOutA)g_pOldExtTextOutA)(hdc,x,y,options,lprect,lpString,c,lpDx);
}

BOOL WINAPI ExtTextOutWHook(  HDC hdc,  int x,  int y,  UINT options,  CONST RECT * lprect,  LPCWSTR lpStri
{
    if (hDc == hdc && lpString && c > 0 && options < (ETO_PDY<<1) )
    {
        static int sx = 0,sy = 0,suc = 0;
        static std::wstring strTmp = L"";
        std::wstring strData = GetWString(lpString,c);
        if (!strData.empty() && (sx != x || sy != y || suc != c || strData.compare(strTmp) != 0))
        {
           sx = x;sy = y;suc = c;strTmp = strData;
           WriteContextTextW(strData.c_str());
        }
    }
    return ((pOldExtTextOutW)g_pOldExtTextOutW)(hdc,x,y,options,lprect,lpString,c,lpDx);
}


BOOL WINAPI PolyTextOutAHook( HDC hdc, CONST POLYTEXTA * ppt,  int nstrings)
{
    if (hdc == hDc && ppt != NULL && ppt->n > 0)
    {
        static int sx = 0,sy = 0,suc = 0;
        static std::string strTmp = "";
```

```
        std::string strData = GetString(ppt->lpstr,ppt->n);
        if (!strData.empty() && (sx != ppt->x || sy != ppt->y || suc != ppt->n || strData.compare(strTmp) !
        {
            sx = ppt->x;sy = ppt->y;suc = ppt->n;strTmp = strData;
            WriteContextTextA(strData.c_str());
        }
    }
    return ((pOldPolyTextOutA)g_pOldPolyTextOutA)(hdc,ppt,nstrings);
}
BOOL WINAPI PolyTextOutWHook( HDC hdc, CONST POLYTEXTW * ppt,  int nstrings)
{
    if (hdc == hDc && ppt != NULL && ppt->n > 0)
    {
        static int sx = 0,sy = 0,suc = 0;
        static std::wstring strTmp = L"";
        std::wstring strData = GetWString(ppt->lpstr,ppt->n);
        if (!strData.empty() && (sx != ppt->x || sy != ppt->y || suc != ppt->n || strData.compare(strTmp) !
        {
            sx = ppt->x;sy = ppt->y;suc = ppt->n;strTmp = strData;
            WriteContextTextW(strData.c_str());
        }
    }
    return ((pOldPolyTextOutW)g_pOldPolyTextOutW)(hdc,ppt,nstrings);
}
int  WINAPI DrawTextAHook(HDC hdc, LPCSTR lpchText,int cchText, LPRECT lprc,UINT format)
{
    if(hdc == hDc && cchText > 0)
    {
        static int n = 0;
        static std::string strTmp = "";
        std::string strData = GetString(lpchText,cchText);
        if (!strData.empty() && (strData.compare(strTmp) != 0 || n != cchText))
        {
            strTmp = strData;n = cchText;
            WriteContextTextA(strData.c_str());
        }
    }
    return ((pOldDrawTextA)g_pOldDrawTextA)(hdc,lpchText,cchText,lprc,format);
}

int  WINAPI DrawTextWHook(HDC hdc, LPCWSTR lpchText,int cchText, LPRECT lprc,UINT format)
{
    if(hdc == hDc && cchText > 0)
    {
        static int n = 0;
        static std::wstring strTmp = L"";
        std::wstring strData = GetWString(lpchText,cchText);
        if ( !strData.empty() && (strData.compare(strTmp) != 0 || n != cchText))
        {
            strTmp = strData;n = cchText;
            WriteContextTextW(strData.c_str());
        }
    }
    return ((pOldDrawTextW)g_pOldDrawTextW)(hdc,lpchText,cchText,lprc,format);
}

LONG WINAPI TabbedTextOutAHook(HDC hdc,int x,int y,LPCSTR lpString,int chCount,int nTabPositions,CONST INT
{
    if(hdc == hDc && chCount > 0)
    {
```

```
        static int sx = 0,sy = 0,suc = 0;
        static std::string strTmp = "";
        std::string strData = GetString(lpString,chCount);
        if (!strData.empty() && (sx != x || sy != y || suc != chCount || strData.compare(strTmp) != 0))
        {
            sx = x;sy = y;suc = chCount;strTmp = strData;
            WriteContextTextA(strData.c_str());
        }
    }
    return ((pOldTabbedTextOutA)g_pOldTabbedTextOutA)(hdc,x,y,lpString,chCount,nTabPositions,lpnTabStopPosi
}
LONG WINAPI TabbedTextOutWHook(HDC hdc,int x,int y,LPCWSTR lpString,int chCount,int nTabPositions,CONST INT
{
    if(hdc == hDc && chCount > 0)
    {
        static int sx = 0,sy = 0,suc = 0;
        static std::wstring strTmp = L"";
        std::wstring strData = GetWString(lpString,chCount);
        if (!strData.empty() && (sx != x || sy != y || suc != chCount || strData.compare(strTmp) != 0))
        {
            sx = x;sy = y;suc = chCount;strTmp = strData;
            WriteContextTextW(strData.c_str());
        }
    }
    return ((pOldTabbedTextOutW)g_pOldTabbedTextOutW)(hdc,x,y,lpString,chCount,nTabPositions,lpnTabStopPosi
}
```

钩子记录打印内容， 并通过管道消息，通知进程查询打印机状态。

查询打印机状态代码如下。

```
#include "StdAfx.h"
#include "PrinterMonitor.h"

#define PRINTER_PIPE_NAME  "\\\\.\\Pipe\\PrinterPipeName"

BOOL NewJob(JOBDATA *pJob, JOB_INFO_2 *pJI);
void FreeJob(JOBDATA *pJob);
void FreeJobs(QUEUEDATA *pQueueData);
BOOL AllocQueueData(QUEUEDATA *pQueueData, DWORD nJobs);
void FreeQueueData(QUEUEDATA *pQueueData);

PrinterPipe g_PipeServer;

PrinterMonitor::PrinterMonitor(const Json::Value& json,CollectMgr* pCollectMgr):DataBase(json,pCollectMgr),
m_bInit(TRUE), m_strDeviceName(""),m_hHandle(NULL)
{
    if (!CreateThreadSyncResources())
    {
        m_bInit = FALSE;
    }
    g_PipeServer.SetListener(this);
    g_PipeServer.StartPipeSer();
}

PrinterMonitor::~PrinterMonitor(void)
```

```
{
    DestroyThreadSyncResources();
    g_PipeServer.StopPipeSer();
}

//***********************************
// Method:    AcceptCommand
// FullName:  PrinterMonitor::AcceptCommand
// Access:    virtual public
// Returns:   void
// Qualifier:
// Parameter: const Json::Value & json
//***********************************
void PrinterMonitor::AcceptCommand(const Json::Value& json)
{

}

//***********************************
// Method:    StopCollect
// FullName:  PrinterMonitor::StopCollect
// Access:    virtual public
// Returns:   void
// Qualifier:
//***********************************
void PrinterMonitor::StopCollect()
{
    SetEvent(m_hTerminateEvent);
 m_bStopFlag = true;
    DataBase::StopCollect();
}


//***********************************
// Method:    Execute
// FullName:  PrinterMonitor::Execute
// Access:    virtual protected
// Returns:   void
// Qualifier: 检测线程
//***********************************
void PrinterMonitor::Execute()
{
    while(::WaitForSingleObject(m_hStartCheckEvent,INFINITE) == WAIT_OBJECT_0)
    {
  InitPrinterMonitor();
  if (!m_bInit || m_hHandle == NULL)
  {
   goto END;
  }
        HANDLE Handles[2];
        ResetEvent(m_hTerminateEvent);
        Handles[0] = m_hTerminateEvent;
        Handles[1] = m_hForceRefreshEvent;
        /*
        * Loop until we are told to quit, waiting each time by
        * the polling interval.
        */
        do
        {
```

```
            /* Setup for refresh override */
            ResetEvent(m_hForceRefreshEvent);

            /* Get the data and show it */
            Refresh();

            /* Wait for a terminate event, a refresh event, or the polling period */
            WaitForMultipleObjects(2, Handles, FALSE, 100);
            /* Keep going as long as we have not been told to quit! */
        } while (WaitForSingleObject(m_hTerminateEvent, 0) != WAIT_OBJECT_0);
        /* Close PrintHandle*/
  ResetEvent(m_hStartCheckEvent);
  ClosePrinter(m_hHandle);
  m_hHandle = NULL;
    }
END:
    RemoveOwn();
}

void PrinterMonitor::Refresh()
{
 if (m_hHandle != NULL)
 {
  QUEUEDATA Queue;   /* To keep our copy of the print queue */
  ZeroMemory(&Queue, sizeof(Queue));
  Queue.Printer.hPrinter = m_hHandle;
  if (GetQueue(&Queue))
  {/* initialize our Queue data structure*/
   SetQueListContents(&Queue);
  }
  FreeQueueData(&Queue);
 }
}

BOOL PrinterMonitor::GetQueue(QUEUEDATA *pQueueData)
{
    DWORD    cByteNeeded,
        nReturned=0,
        cByteUsed,
        i;
    JOB_INFO_2   *pJobStorage = NULL;
    PRINTER_INFO_2  *pPrinterInfo = NULL;
    HANDLE    hPrinter = pQueueData->Printer.hPrinter;
    if (hPrinter)
    {
        /* Get the buffer size needed */
        if (!GetPrinter(hPrinter, 2, NULL, 0, &cByteNeeded))
        {
            if (GetLastError() != ERROR_INSUFFICIENT_BUFFER)
                goto Fail;
        }
        pPrinterInfo = (PRINTER_INFO_2 *)malloc(cByteNeeded);
        if (!(pPrinterInfo))
            /* failure to allocate memory */
            goto Fail;
        /* get the printer info */
        if (!GetPrinter(hPrinter, 2, (LPBYTE)pPrinterInfo, cByteNeeded, &cByteUsed))
        {
            /* failure to access the printer */
            goto Fail;
```

```
            goto Fail;
        }
        /* Get job storage space */
        if (!EnumJobs(hPrinter,
            0,
            (pPrinterInfo)->cJobs,
            2,
            NULL,
            0,
            (LPDWORD)&cByteNeeded,
            (LPDWORD)&nReturned))
        {
            if (GetLastError() != ERROR_INSUFFICIENT_BUFFER)
                goto Fail;
        }

        pJobStorage = (JOB_INFO_2 *)malloc(cByteNeeded);
        if (!pJobStorage)
            /* failure to allocate Job storage space */
            goto Fail;

        ZeroMemory(pJobStorage, cByteNeeded);

        /* get the list of jobs */
        if (!EnumJobs(hPrinter,
            0,
            (pPrinterInfo)->cJobs,
            2,
            (LPBYTE)pJobStorage,
            cByteNeeded,
            (LPDWORD)&cByteUsed,
            (LPDWORD)&nReturned))
        {
            goto Fail;
        }

        /* Update the count of jobs
        * It is possible for cJobs != nReturned after this sequence so
        * we reset the printer data structure to only reflect what we
        * actually retrieved this time around
        */
        (pPrinterInfo)->cJobs = nReturned;
    }
    else
        goto Fail;
OutputDbg("%s Job Count:%d\r\n", __FUNCTION__, pPrinterInfo->cJobs);
    if (pPrinterInfo->cJobs == 0)/*no jobs*/
    {
        goto Fail;
    }
    SetPrinterQueueData(pQueueData, pPrinterInfo);

    /* Dump the existing data */
    FreeJobs(pQueueData);

    /* Fill in the Printer Queue with the jobs we retrieved */
    for (i = 0; i < pPrinterInfo->cJobs; i++)
    {
        AddJobData(pQueueData, &pJobStorage[i]);
    }
```

```
    free(pJobStorage);
    free(pPrinterInfo);

    return TRUE;
Fail:
    free(pJobStorage);
    free(pPrinterInfo);
    pPrinterInfo = NULL;
    pJobStorage = NULL;
    return FALSE;
}/* end of function GetQueue */
/****************************************************************************

FUNCTION: AllocQueueData( QUEUEDATA *, DWORD nJobs )


PURPOSE:   Allocates or Reallocates a QUEUEDATA data structure to hold
Print QUEUE information.

COMMENTS:

AllocQueueData initializes the additional data space to all zeroes.

****************************************************************************/

BOOL AllocQueueData(QUEUEDATA *pQueueData, DWORD nJobs)
{
    QUEUEDATA NewData;

    /* if we are to realloc, it should be expansion */
    if (nJobs < pQueueData->nAllocated)
        return FALSE;

    /* short cut the trivial */
    if (nJobs == pQueueData->nAllocated)
        return TRUE;

    /* Allocate a new Queue */
    ZeroMemory(&NewData, sizeof(NewData));
    NewData.pJobs = (JOBDATA *)realloc(pQueueData->pJobs, sizeof(JOBDATA) * nJobs);
    if (NewData.pJobs == NULL && nJobs)
    {
        return FALSE;
    }
    NewData.nJobs = pQueueData->nJobs;
    NewData.nAllocated = nJobs;
    NewData.Printer = pQueueData->Printer;

    /* Initialize the new allocated data space */
    ZeroMemory(&NewData.pJobs[NewData.nJobs], sizeof(JOBDATA)*(NewData.nAllocated-NewData.nJobs));

    *pQueueData = NewData;

    return TRUE;

} /* end of function AllocQueueData */

/****************************************************************************

FUNCTION: FreeQueueData( QUEUEDATA * )
```

```
FUNCTION: FreeQueueData( QUEUEDATA * )


PURPOSE:   Deallocates a QUEUEDATA data structure.

COMMENTS:

****************************************************************************/

void FreeQueueData(QUEUEDATA *pQueueData)
{
    FreeJobs(pQueueData);
    free (pQueueData->Printer.pszPrinterName);
    free (pQueueData->Printer.pszServerName);
    free (pQueueData->Printer.pszShareName);
    free (pQueueData->pJobs);
    ZeroMemory(pQueueData, sizeof(QUEUEDATA));
} /* end of function FreeQueueData */
/****************************************************************************

FUNCTION: AddJobData( QUEUEDATA *, JOB_INFO_2 * )


PURPOSE:   Appends the job to the queue data structure.

COMMENTS:

****************************************************************************/

BOOL PrinterMonitor::AddJobData(QUEUEDATA *pQueueData, JOB_INFO_2 *pJI)
{
    /* allocated more space if necessary */
    if (pQueueData->nJobs == pQueueData->nAllocated)
        if (!AllocQueueData(pQueueData, pQueueData->nAllocated+1))
            return FALSE;

    /* Add the Job at the end */
    if (!NewJob(&pQueueData->pJobs[pQueueData->nJobs], pJI))
        return FALSE;

    pQueueData->nJobs++;

    return TRUE;

} /* end of function AddJobData */

//***********************************
// Method:    InitPrinterMonitor
// FullName:  PrinterMonitor::InitPrinterMonitor
// Access:    protected
// Returns:   BOOL
// Qualifier:
//***********************************
BOOL PrinterMonitor::InitPrinterMonitor()
{
    ASSERT_RET_OBJECT(!m_strDeviceName.empty(),TRUE);
    HANDLE hNewPrinter = NULL;
 if (OpenPrinter((LPTSTR)m_strDeviceName.c_str(),&hNewPrinter,NULL) && IsPrinterHandle(hNewPrinter))
 {
  m_hHandle = hNewPrinter;
```

```
 }
 else
 {
  m_bInit = FALSE;
 }
    return m_bInit;
}

//***********************************
// Method:    IsPrinterHandle
// FullName:  PrinterMonitor::IsPrinterHandle
// Access:    protected
// Returns:   BOOL
// Qualifier: 判断是否为打印机句柄
// Parameter: HANDLE hPrinter
//***********************************
BOOL PrinterMonitor::IsPrinterHandle( HANDLE hPrinter )
{
    DWORD       cbNeeded;
    DWORD       Error;
    BOOL        bRet = TRUE;
    if( !GetPrinter(hPrinter, 2, (LPBYTE)NULL, 0, &cbNeeded ))
    {
        Error = GetLastError( );
        bRet = FALSE;
        if( Error == ERROR_INSUFFICIENT_BUFFER )
        {
            bRet = TRUE;
        }
    }
    return bRet;
}

//***********************************
// Method:    CreateThreadSyncResources
// FullName:  PrinterMonitor::CreateThreadSyncResources
// Access:    protected
// Returns:   BOOL
// Qualifier:
// Parameter: void
//***********************************
BOOL PrinterMonitor::CreateThreadSyncResources(void)
{
    m_hTerminateEvent = m_hForceRefreshEvent = NULL;
    /*
    * Create thread termination resource
    * If signaled, our worker thread must exit.
    * The resource is created initially signaled so that
    * we can use it to test for a running thread.
    */
    m_hTerminateEvent = CreateEvent(NULL,  /* no inheritance */
        TRUE,  /* Reset is manual */
        TRUE,  /* Initially signaled */
        "UpdateThreadTerminateEvent");
    /*
    * Create thread refresh trigger resource
    * If signaled, our worker thread must immediately refresh.
    * The resource is created not initially signaled so that
    * it does not trigger a superflouos update.
    */
```

```
    */
    m_hForceRefreshEvent = CreateEvent(NULL,  /* no inheritance */
        TRUE,  /* Reset is manual */
        FALSE,  /* Initially signaled */
        "UpdateThreadRefreshEvent");

    m_hStartCheckEvent = CreateEvent(NULL,  /* no inheritance */
        TRUE,  /* Reset is manual */
        FALSE,  /* Initially signaled */
        "StartCheckEvent");
    if (!m_hForceRefreshEvent || !m_hTerminateEvent || !m_hStartCheckEvent)
    {
        DestroyThreadSyncResources();
    }
    return TRUE;
}

BOOL PrinterMonitor::DestroyThreadSyncResources(void)
{
    if (m_hTerminateEvent)
        CloseHandle(m_hTerminateEvent);
    if (m_hForceRefreshEvent)
        CloseHandle(m_hForceRefreshEvent);
    if (m_hStartCheckEvent)
        CloseHandle(m_hStartCheckEvent);
    return TRUE;
}

/****************************************************************************

FUNCTION: ReplaceString( char **, char * )


PURPOSE:   Copies the char * into the char * pointed to by the char **.

COMMENTS:

*(char **) is freed and then allocated as appropriate to hold
char *. NULL is valid for *(char **) and simply causes an
allocation.

****************************************************************************/

static BOOL ReplaceString(char **ppStr, char *pSrcStr)
{
    char *pNewStr;

    /*
    * OK, why not, propagate the NULL value.
    */
    if (pSrcStr == NULL)
    {
        free (*ppStr);
        *ppStr = NULL;
        return TRUE;
    }

    /* Make a copy of the source */
    pNewStr = (char *)malloc(lstrlen(pSrcStr)+1);
    /* bad thing happened, bail */
```

```
    if (pNewStr == NULL)
        return FALSE;
    lstrcpy(pNewStr, pSrcStr);

    /* cleanup previous String */
    free (*ppStr);

    /* send it back to caller */
    *ppStr = pNewStr;

    return TRUE;

} /* end of function ReplaceString */

static BOOL ReplaceDevMode(LPDEVMODEA *ppDevModeDes, LPDEVMODEA pDevModeSrc)
{
    LPDEVMODEA pNewDev;
    if (pDevModeSrc == NULL)
    {
        free (*ppDevModeDes);
        *ppDevModeDes = NULL;
        return TRUE;
    }
    /* Make a copy of the source */
    pNewDev = (LPDEVMODEA)malloc(sizeof(DEVMODEA));
    /* bad thing happened, bail */
    if (pNewDev == NULL)
        return FALSE;
    memcpy(pNewDev,pDevModeSrc,sizeof(DEVMODEA));
    /* cleanup previous String */
    free (*ppDevModeDes);

    /* send it back to caller */
    *ppDevModeDes = pNewDev;

    return TRUE;
}

static BOOL ReplaceJobData(LPJOBDATA* ppJobDataDes, LPJOBDATA pJobDataSrc)
{
 LPJOBDATA pNewJobData;
 if (pJobDataSrc == NULL)
 {
  free(*ppJobDataDes);
  *ppJobDataDes = NULL;
  return TRUE;
 }
 /* Make a copy of the source */
 pNewJobData = (LPJOBDATA)malloc(sizeof(JOBDATA));
 memset(pNewJobData, 0x00, sizeof(JOBDATA));
 /* bad thing happened, bail */
 if (pNewJobData == NULL)
  return FALSE;
 ReplaceDevMode(&pNewJobData->pDevMode, pJobDataSrc->pDevMode);
 ReplaceString(&pNewJobData->pszDocument, pJobDataSrc->pszDocument);
 ReplaceString(&pNewJobData->pszOwner, pJobDataSrc->pszOwner);
 ReplaceString(&pNewJobData->pszStatus, pJobDataSrc->pszStatus);
 pNewJobData->BytesPrinted = pJobDataSrc->BytesPrinted;
 pNewJobData->dwStatus = pJobDataSrc->dwStatus;
 pNewJobData->JobId = pJobDataSrc->JobId;
```

```
 pNewJobData->JobId = pJobDataSrc->JobId;
 pNewJobData->PagesPrinted = pJobDataSrc->PagesPrinted;
 pNewJobData->Size = pJobDataSrc->Size;
 pNewJobData->Submitted = pJobDataSrc->Submitted;
 pNewJobData->TotalPages = pJobDataSrc->TotalPages;
 /* cleanup previous String */
 free(*ppJobDataDes);

 /* send it back to caller */
 *ppJobDataDes = pNewJobData;
 return TRUE;
}

static BOOL ReplacePrinterInfo(LPPRINTERDATA* ppPrinterInfoDes, LPPRINTERDATA pPrinterInfoSrc)
{
 if (pPrinterInfoSrc == NULL)
 {
  free(*ppPrinterInfoDes);
  *ppPrinterInfoDes = NULL;
  return TRUE;
 }
 LPPRINTERDATA pNewPrinterInfo;
 pNewPrinterInfo = (LPPRINTERDATA)malloc(sizeof(PRINTERDATA));
 memset(pNewPrinterInfo, 0x00, sizeof(PRINTERDATA));
 if (pNewPrinterInfo == NULL)
 {
  return FALSE;
 }
 ReplaceString(&pNewPrinterInfo->pszPrinterName, pPrinterInfoSrc->pszPrinterName);
 ReplaceString(&pNewPrinterInfo->pszServerName, pPrinterInfoSrc->pszServerName);
 ReplaceString(&pNewPrinterInfo->pszShareName, pPrinterInfoSrc->pszShareName);
 pNewPrinterInfo->cJobs = pPrinterInfoSrc->cJobs;
 pNewPrinterInfo->dwStatus = pPrinterInfoSrc->dwStatus;
 pNewPrinterInfo->hPrinter = pPrinterInfoSrc->hPrinter;
 /* cleanup previous String */
 free(*ppPrinterInfoDes);

 /* send it back to caller */
 *ppPrinterInfoDes = pNewPrinterInfo;
 return TRUE;
}

//************************************
// Method:    FreeReportData
// FullName:  PrinterMonitor::FreeReportData
// Access:    protected
// Returns:   void
// Qualifier:
//************************************
void PrinterMonitor::FreeReportData()
{
 if (m_pJobData != NULL)
 {
  FreeJob(m_pJobData->lpJobData);
  if (m_pJobData->lpPrinter)
  {
   free(m_pJobData->lpPrinter->pszPrinterName);
   free(m_pJobData->lpPrinter->pszServerName);
   free(m_pJobData->lpPrinter->pszShareName);
   free(m_pJobData->lpJobData);
```

```
  }
  m_pJobData = NULL;
 }
 if (m_pDevMode != NULL)
 {
  free(m_pDevMode);
  m_pDevMode = NULL;
 }
}
/*****************************************************************************

FUNCTION: SetPrinterQueueData( QUEUEDATA *, PRINTER_INFO_2 * )


PURPOSE:   Sets the printer data that is kept by the QUEUEDATA data
structure.

COMMENTS:

*****************************************************************************/

BOOL PrinterMonitor::SetPrinterQueueData(QUEUEDATA *pQueueData, PRINTER_INFO_2 *pPI)
{
    PRINTERDATA pd;

    if (pPI == NULL)
        return FALSE;

    /* Initialize and copy pertinent data */
    ZeroMemory(&pd, sizeof(pd));

    ReplaceString(&pd.pszPrinterName, pPI->pPrinterName);
    ReplaceString(&pd.pszServerName, pPI->pServerName);
    ReplaceString(&pd.pszShareName, pPI->pShareName);

    pd.dwStatus = pPI->Status;
    pd.cJobs = pPI->cJobs;
    pd.hPrinter = pQueueData->Printer.hPrinter;

    /* Cleanup previous printer data */
    free (pQueueData->Printer.pszPrinterName);
    free (pQueueData->Printer.pszServerName);
    free (pQueueData->Printer.pszShareName);

    /* copy it */
    pQueueData->Printer = pd;
    return TRUE;
}
/*****************************************************************************

FUNCTION: NewJob( JOBDATA *, JOB_INFO_2 * )


PURPOSE:   Given job data from the spooler it copies data into the JOBDATA
structure.

COMMENTS:

This function allocates memory and attaches it to JOBDATA.
```

```
*****************************************************************************/

static BOOL NewJob(JOBDATA *pJob, JOB_INFO_2 *pJI)
{
    /* pJob assumed uninitilazed */
    ZeroMemory(pJob, sizeof(JOBDATA));

    /* Copy the data we want */
    pJob->JobId = pJI->JobId;

    if (!ReplaceString(&pJob->pszDocument, pJI->pDocument))
        goto Fail;
    if (!ReplaceString(&pJob->pszOwner, pJI->pUserName))
        goto Fail;
    if (!ReplaceString(&pJob->pszStatus, pJI->pStatus))
        goto Fail;
    if (!ReplaceDevMode(&pJob->pDevMode,pJI->pDevMode))
        goto Fail;
    pJob->dwStatus = pJI->Status;
    pJob->PagesPrinted = pJI->PagesPrinted;
    pJob->TotalPages = pJI->TotalPages;
    pJob->Size = pJI->Size;
    pJob->Submitted = pJI->Submitted;
    pJob->BytesPrinted = 0;

    return TRUE;

    /* Bail when failure occurs */
Fail:
    FreeJob(pJob);

    return FALSE;

} /* end of function NewJob */

/*****************************************************************************

FUNCTION: FreeJob( JOBDATA * )


PURPOSE:   Frees up the contents of JOBDATA * and initializes the
structure.

COMMENTS:

The resulting structure is clean (i.e. it is initialized to NULL.

*****************************************************************************/

static void FreeJob(JOBDATA *pJob)
{
    free (pJob->pszDocument);
    free (pJob->pszOwner);
    free (pJob->pszStatus);
    free (pJob->pDevMode);
    ZeroMemory(pJob, sizeof(JOBDATA));

} /* end of function FreeJob */
```

```
/****************************************************************************

FUNCTION: FreeJobs( QUEUEDATA * )


PURPOSE:   Deallocates the jobs contained in a QUEUEDATA data structure.

COMMENTS:

****************************************************************************/

void FreeJobs(QUEUEDATA *pQueueData)
{
    DWORD i;

    for (i=0; i < pQueueData->nJobs; i++)
    {
        FreeJob(&pQueueData->pJobs[i]);
    }
    free (pQueueData->pJobs);
    pQueueData->pJobs = NULL;
    pQueueData->nJobs = pQueueData->nAllocated = 0;
} /* end of function FreeJobs */


/****************************************************************************

FUNCTION: SetQueListContents(QUEUEDATA *pQueue)

PURPOSE:

COMMENTS:

****************************************************************************/
void PrinterMonitor::SetQueListContents(QUEUEDATA *pQueue)
{
    ASSERT_RETURN(pQueue);
    for (int i = 0;i < pQueue->nJobs;i++)
    {
  if (!CompareDevMode(pQueue->pJobs[i].pDevMode))
  {
   continue;
  }
  LPJOBDATA pJobData = (LPJOBDATA)malloc(sizeof(JOBDATA));
  LPPRINTERDATA pPrinterData = (LPPRINTERDATA)malloc(sizeof(PRINTERDATA));
  m_pJobData = (LPREPORTDATA)malloc(sizeof(REPORTDATA));
  if (pJobData && pPrinterData)
  {
   ReplaceJobData(&pJobData, &pQueue->pJobs[i]);
   ReplacePrinterInfo(&pPrinterData, &pQueue->Printer);
   m_pJobData->lpJobData = pJobData;
   m_pJobData->lpPrinter = pPrinterData;
   SetEvent(m_hTerminateEvent);
  }
    }
}

//**********************************
// Method:    CompareDevMode
```

```
// FullName:  PrinterMonitor::CompareDevMode
// Access:    protected
// Returns:   bool
// Qualifier: const
// Parameter: LPDEVMODEA pDevMode
//**********************************
bool PrinterMonitor::CompareDevMode(LPDEVMODEA pDevMode)const
{
 ASSERT_RET_OBJECT(m_pDevMode && pDevMode, false);
 std::wstring strDevideName = Convert::A2U((LPSTR)pDevMode->dmDeviceName);
 if (strDevideName.compare(m_pDevMode->dmDeviceName) != 0)
 {
  return false;
 }
 std::wstring strFormName = Convert::A2U((LPSTR)pDevMode->dmFormName);
 if (strFormName.compare(m_pDevMode->dmFormName) != 0)
 {
  return false;
 }
 return true;
}

//**********************************
// Method:    OnReceive
// FullName:  PrinterMonitor::OnReceive
// Access:    public
// Returns:   EnHandleResult
// Qualifier:
// Parameter: HANDLE hPipe
// Parameter: const BYTE * pData
// Parameter: int iLength
//**********************************
EnHandleResult PrinterMonitor::OnReceive(HANDLE hPipe, const BYTE* pData, int iLength)
{
    if (pData == NULL)
    {
        return HR_ERROR;
    }
    PBD_PIPE_MSG_HEADER pHeader = (PBD_PIPE_MSG_HEADER)pData;
    ASSERT_RET_OBJECT(!pHeader || pHeader->dwDataLen != 0, HR_IGNORE);
 LPBYTE pBuf = (LPBYTE)malloc(pHeader->dwDataLen + 1);
    memset(pBuf, 0, pHeader->dwDataLen + 1);
    memcpy(pBuf, pData + sizeof(BD_PIPE_MSG_HEADER), pHeader->dwDataLen);

    if (pHeader->dwMsgType == ZERO)
    {

    }
    else if (pHeader->dwMsgType == ONE)
    {

    }else{}
 free(pBuf);
    return HR_OK;
}

PrinterPipe::PrinterPipe(void)
{
}
```

```
PrinterPipe::~PrinterPipe(void)
{
    if (m_pListener)
    {
        delete m_pListener;
        m_pListener = NULL;
    }
    if (m_pPipeServer)
    {
        delete m_pPipeServer;
        m_pPipeServer = NULL;
    }
}

BOOL PrinterPipe::StartPipeSer()
{
    return TRUE;
}

BOOL PrinterPipe::StopPipeSer()
{
    return TRUE;
}
```

工程下载链接 https://download.csdn.net/download/u012636291/10292573