Extending Procomm Plus® Using Dynamically Linked Protocols

Table of Contents

INTRODUCTION 2

DLP OVERVIEW 2

AN OVERVIEW OF PROCOMM PLUS CALLBACK FUNCTIONS 3

Access To Windows SDK Comm Functions 3

Procomm Plus Functions Specifically Intended to Support DLPs 3

XMODEM.DLP - A SAMPLE DLP 6

Multitasking and Reentrancy Considerations 6

Program Components: Building the Sample DLP 7

DATA STRUCTURES 9

A CLOSER LOOK AT DLP ENTRY POINTS 9

ADVANCED SETUP ENTRY POINTS 11

INDEX OF FUNCTIONS 13

Introduction

See the document ***Procomm Plus® Interface Specification Overview and Shared Callback Functions*** for introductory information and callback functions which can be used by any DL*x*.

DLP Overview

A DLP is a custom file transfer protocol written as a standard Windows DLL. It has a **DllMain**() and library entry points that are defined as exported functions in the .DEF file. All DLPs must have certain standard entry points; Procomm Plus itself makes many of its services available to the DLP through exported callback procedures.

> ⇒ ***Note:*** *This document documents the DLP and its entry points; you should also obtain* **Procomm Plus® Interface Specification Overview and Shared Callback Functions***, which describes Procomm Plus's callback functions.*

DLPs have a set of standard library entry points, which are summarized here:

GET_DLL_CAPS () - Called by Procomm Plus at the start of a file transfer session. It retrieves attribute flags from the DLP, which indicate whether certain tasks related to file transfers should be performed by Procomm Plus. For example, whether Procomm Plus should watch for autodownload strings in the data stream or whether Procomm Plus must ask the DLP to display an Advanced Settings Dialog.

STARTDLL_XFER () - Called by Procomm Plus to initiate the file transfer.

DLL_XFER () - Called periodically by Procomm Plus (approximately 18 times / sec.). The DLP can utilize the **DLL_XFER** call as a rudimentary multitasking state-switcher while the file transfer is in progress.

INITDLGITEMS () - Called by Procomm Plus prior to displaying the Advanced Settings Dialog Box. **InitDlgItems()** should initialize the dialog contents according to the protocol's default values.

SETWM_COMMAND () - Procomm Plus inserts a hook into the Advanced Settings Dialog Box processing, allowing the DLP to perform any special processing when a related dialog control is selected or altered.

SETDEFAULTS () - Called by Procomm Plus when the user selects "Use Defaults" from the "Modify Option Set" dialog accessed via the "Transfer Protocol" setup dialog. The various data fields should be reset to their default values.

GETDLGITEMS () - Called by Procomm Plus when the Advanced Settings Dialog Box is closed by the user. The dialog's various field and checkbox values are moved into their appropriate data structure locations.

An Overview of Procomm Plus Callback Functions

Access To Windows SDK Comm Functions

Procomm Plus renames and exports each of the nine standard Windows Comm Functions for access by a DL*x*. Their definitions are more fully explained in ***Procomm Plus for Windows™ Interface Specification Overview and Shared Callback Functions*****.**

Procomm Plus Functions Specifically Intended to Support DLPs

In addition to the standard Windows Comm Functions, Procomm Plus also exports the following functions, which are of particular importance to DLP authors:

EXPORTED BOOL CALLBACK P_FileRxTX(int rcv, LPSTR ptr)

rcv should be 0 if sending a file and non-zero if receiving a file. If non-zero the file will be created if necessary.

ptr should point to a null terminated string containing the filename to open. It must not contain any path information. The routine will use the path identified in **pd.szTempDir.**

Should be called by the DLP to open a file. Procomm Plus opens the file, handles any error conditions associated with opening the file, and updates the following fields in the **pd** data structure:

pd.hCommFile - receives the file handle once the file is opened or -1 if no file is currently open.

pd.flen - contains the length (in bytes) of the current file being transferred.

pd.fdate - contains the date the file was last updated (DOS file date format) or 0 if the file is being received.

pd.fftime - contains the time the file was last updated (DOS file time format) or 0 if the file is being received.

pd.szFullPath - contains the full path and filename of the current file or NULL if no file is currently open.

As stated earlier, the file will be opened in the subdirectory identified in **pd.szTempDir**, and the function will build **pd.szFullPath**.

⇒ ***Note:*** *Do not call this function to open a file if you are also setting the DLL Capabilities bit requesting Procomm Plus to automatically open the file for you. See the description of the* ***GET_DLL_CAPS*** *entry point for further details.*

EXPORTED void CALLBACK P_XferSuccess (void)

Should be called by the DLP to indicate a successful file transfer. This should be called before **P_EndXfer().**

EXPORTED void CALLBACK P_XferFail (void)

Should be called by the DLP to indicate a failed file transfer. This should be called before **P_EndXfer().**

EXPORTED void CALLBACK P_EndXfer (void)

Should be called by the DLP to exit file transfer mode and return to terminal emulation mode. If the file named by the **pd.hCommFile** field is still open, Procomm Plus will close it. If the file being closed was received, Procomm Plus sets the file's date and time stamp. If supporting batch transfers you should only call this routine after the last file has been transferred. You should also close the file named in **pd.hCommFile** before calling **P_FileRxTX()** to open the next file in the batch.

EXPORTED void CALLBACK P_ShowXferBox (int send)

send should be non-zero if sending and zero when receiving.

If the DLL_CAP_SHOWXFER flag is NOT used in the **GET_DLL_CAPS()** routine of the .DLP then **P_ShowXferBox** should be called once the file transfer is underway. This function notifies Procomm Plus to display the standard file transfer status dialog box. Prior to calling this function, the following fields need to be set to appropriate values:

pd.bdiag - should be non-zero to enable the display of the file transfer status dialog box.

pd.flen - should equal the length (in bytes) of the file currently being transferred, or 0 if the length is unknown.

pd.runcnt - should equal the number of bytes transferred thus far.

EXPORTED void CALLBACK P_TimeToGo (void)

Should be called periodically by the DLP while the standard file transfer status dialog box is displayed. This function notifies Procomm Plus to display updated status fields in the file-transfer dialog box. The DLP should update the following field before calling this function:

pd.runcnt - should equal the current total number of bytes transferred.

EXPORTED void CALLBACK P_LastError (int i)

i should be set to the index in the common error string table.

Should be called by the DLP when it detects an error in the file transfer. This function notifies Procomm Plus to display an appropriate error message in the file transfer status dialog box.

⇒ ***Note:*** *A list of supported error messages is not provided in this document. Write a test routine to determine if there are any messages useful to the DLP. If not, simply update the transfer dialog error field directly.*

EXPORTED void CALLBACK P_StartGif (void)

If the DLP determines that it is receiving a .GIF file, and the user wants to view the .GIF as it is received, the DLP should call this function, advising Procomm Plus to initialize the GIF Viewer.

EXPORTED BOOL CALLBACK P_FeedDisplay (BYTE ch)

ch contains the byte to be displayed.

Should be called by the DLP as each byte of a .GIF file is received, notifying Procomm Plus to update the display.

EXPORTED void CALLBACK P_DeletePartialFile (void)

In the event that a file transfer operation prematurely terminates, what should the receiving system do about the partial file it has written to disk? If the protocol provides a mechanism for restarting a file transfer at any point in the file, you may wish to keep the good portion of the file which has already been received. In most cases, however, it is desirable to delete the partial file. If a DLP simply calls **P_EndXfer**, the partial file will be closed and saved. Calling **P_DeletePartialFile** insures that the file identified in **pd.szFullPath** is deleted.

XMODEM.DLP - a Sample DLP

To illustrate the construction of a Dynamically Linked Protocol, source files for **xmodem.dlp** are provided. In the discussions that follow, assume that **xmodem.dlp** is being used instead of Procomm Plus's built-in XMODEM protocol.

Multitasking and Reentrancy Considerations

Suppose you wanted to download a file using XMODEM on COM1 in one instance of Procomm Plus, and upload another file using XMODEM on COM2 in a second instance of Procomm Plus. Only one instance of the DLP will be loaded, but it must be capable of simultaneously uploading and downloading. How can this be done?

DLPs must be optimized to run well in a multitasking environment. Tasks should be designed to be small, quickly-executing, discrete events. The objective is to execute a small segment of the protocol quickly, then return control to Procomm Plus so that other tasks can run. **xmodem.dlp** uses a simple state-machine to implement the XMODEM file transfer protocol.

When sending a file, the XMODEM protocol builds 128-byte blocks, transmits them to the receiving side, then waits for the receiver to respond with either an ACK or NACK. The **xmodem.dlp** builds a block, transmits it, and returns to Procomm Plus to let other tasks run, instead of wasting processor time while waiting for the receiver's response.

Every timer-tick, Procomm Plus calls into the DLP's "**DLL_XFER**" entry point, causing the appropriate state-execution routine to be dispatched. When sending a file, the "**DLL_XFER**" state-machine calls the "**WAITACK**" routine. "**WAITACK**" checks the receive buffer to determine whether the other side has sent an ACK or NACK. If not, "**WAITACK**" returns to "**DLL_XFER**", which in turn returns to Procomm Plus. When an ACK has been received, the **WAITACK**" routine will call the "**XMODEM_TX**" routine to transmit the next block, and the cycle repeats until the entire file has been transmitted.

When **xmodem.dlp** is receiving a file, the process is quite similar. On every timer tick, Procomm Plus calls the "**DLL_XFER**" entry point. The receive-processing states alternate **RXING**" and "**NOCHAR**". "**RXING**" attempts to read 128 bytes from the Rx Buffer, validate the data received, and transmit either an ACK or NACK. As soon as a response is transmitted, if any, the "**RXING**" routine returns to Procomm Plus. If there aren't enough bytes in the Rx Buffer to assemble a full 128-byte block, the state is changed to "**NOCHAR**" and the DLP returns to Procomm Plus. On each subsequent timer tick, Procomm Plus again calls into the DLP. Eventually, the "**NOCHAR**" routine will be able to complete the 128-byte block. The next processing state is set to "**RXING**" and the DLP returns control to Procomm Plus. This cycle repeats until the entire file has been received.

⇒ ***Note:*** *For simplicity's sake, this discussion ignores the TIMEOUT and ERROR states.*

A DLP must be able to support multiple simultaneous file transfers. In our example above, it is quite likely that on one timer tick, one instance of Procomm Plus will ask **xmodem.dlp** to send the next block of its file, and on the next timer tick, the second instance of Procomm Plus might ask **xmodem.dlp** to receive the next block of a second file. This leads to an interesting question: where is the data saved?

The answer is that each instance of Procomm Plus allocates and locks a block of memory for use by the DLP. This data area is defined by the **pd** data structure. Within the **pd** data structure, there is a 66-byte work area (labeled "temp"), which the DLP can use to store instance-specific data. When the DLP is called, a pointer to the **pd** structure is passed to the DLP, allowing it to store its data at **pd->temp**.

⇒ ***Note:*** *There* is *a tiny bit of global memory defined in* ***xmodem.dlp****, but it is strictly intended for data unrelated to a particular instance of Procomm Plus.*

You should now be able to understand how a DLP is able to handle our example of simultaneous access by two instances of Procomm Plus. Each instance of Procomm Plus starts a file transfer, thereafter calling the DLP's state-machine entry point on every timer tick while the file transfer is ongoing. Each time the DLP is called, it performs a small piece of the file transfer, then saves its state in the caller's block of memory. Thus, other instances of Procomm Plus can call the DLP without fear of corrupting another task's data.

Program Components: Building the Sample DLP

You should have received a set of files with this document. These files contain sample code used to build the simplest of all possible DLPs - an XMODEM file transfer protocol. We chose

XMODEM because we wanted to focus on the DLP mechanism itself and minimize any obfuscation which might result from using a more complex protocol. The following files are necessary to build the sample DLP:

makefile.dlp The input file to Microsoft's NMAKE.

xmodem.c The main source code module for XMODEM.DLP.

xmodem.h An XMODEM-specific include file containing data structures and function prototypes.

cbackdlx.h Definition file containing TYPEDEFS and function prototypes for all Procomm Plus callback functions.

pwdlx.h Definition file containing standard Procomm Plus structures, #defines, and TYPEDEFS used by DLPs.

pwdefdlx.h Definition file containing DLP-specific structures, #defines, and TYPEDEFS.

xmodem.rc Input file for Microsoft's Resource Compiler. It contains a table of all strings displayed as messages by the DLP and defines the Advanced Setup Dialog Box.

xmodem.def Input file for Microsoft's Linker. It lists the exported DLP entry points.

xmodem.lnk Input response file for Microsoft's Linker.

You will also need the following products:

- Microsoft ® 32-bit C/C++ Optimizing Compiler Version 10.20 or later.
- Microsoft ® 32-Bit Incremental Linker Version 4.20 or later.
- Microsoft ® Program Maintenance Utility Version 1.6 or later.

Install these files on your system and invoke the NMAKE program; it will build **xmodem.dlp**. Then, simply copy **xmodem.dlp** to the same subdirectory where you installed Procomm Plus.

Data Structures

Very little data is defined as global memory in a DL*x*. Since multiple instances of Procomm Plus may interleave their accesses to a DL*x*, you should not use global data items to store data unique to a specific instance. In the sample **xmodem.c**, several pointers are defined in global memory, but a study of the code will reveal that these pointers are reloaded each time one of the DLP's entry points is called. The values used to reload these pointers are passed in by Procomm Plus when it calls the DL*x*. The *only* reason they are globals is that it is *slightly* more efficient to refer to global variables (which by default are accessed via the DS register) than to refer to local variables (which have the BP register override). Admittedly, this is a minor tweak; some programmers believe that global variables should be avoided, and may choose to use local variables. We have included this example to illustrate one way to increase file transfer speed by a few CPS.

Most of the data used by **xmodem.c** reside in two memory blocks allocated by each instance of Procomm Plus. Most of the general options used by Procomm Plus and a DLP to control the file transfer are defined in a structure named ***pd***. The ***pd*** data structure is defined in **pwdefdlx.h** with the **PROTOSET** typedef. When Procomm Plus calls either the **STARTDLL_XFER**() or the **DLL_XFER**() entry point in a DLP, it passes a pointer to its ***pd*** data structure. Contained in the block defined by the ***pd*** data structure are 66 bytes labeled ***temp***, which are available for any purpose by a DLP. **Xmodem.dlp** uses some of the ***temp*** area to save instance-specific data. This data area is temporary, but usable throughout the current protocol session. For an example, see the **INSTANCEDATA** structure in **xmodem.h.**

The second major data structure shared by Procomm Plus and a DLP is defined in **xmodem.h** with the **ADVANCEDSET** typedef. This is a 325-byte block of memory allocated by Procomm Plus, intended to be used to save data related to a protocol's Advanced Setup options. All 325 bytes are restored from disk when Procomm Plus is started, and saved to disk when Procomm Plus Setup is exited with the OK push button. A pointer to this memory block is passed as an argument in several of the DLP entry points.

> ⇒ ***HINT:*** *If your Advanced Setup options don't need all 325 bytes, the unused area is available to your DLP for storing additional instance-specific data. However, since this area is saved and restored, you may need to explicitly initialize any temporary data fields you define in this area. Keep in mind that each active instance of Procomm Plus can cause this data area to be saved to disk, so treat the use of this area as temporary for the current protocol session. An appropriate time to perform initialization of this temporary memory would be at the top of* ***STARTDLL_XFER****().*

Message strings are defined in the resource file, **xmodem.rc**. To facilitate the international versions of Procomm Plus, all strings were moved from the .C and .H files into the .RC files, where they are defined as indexed items in **STRINGTABLE**.

A Closer Look at DLP Entry Points

Xmodem.c demonstrates the bulk of the DLP. As mentioned earlier, all DLPs have a set of standard entry points of interest to developers. There is also one other standard entry point that we can dispense with fairly quickly:

int WINAPI **DllMain (**HINSTANCE hInstance, DWORD dwReason,
LPVOID lpReserved**)**

When Procomm Plus requests Windows to load a DL*x*, Windows will always call the **DllMain** procedure in the DL*x*. However, the only real processing done by **DllMain** is to save the DLx's Instance Handle.

The remaining entry points are much more important to a DLP:

WORD CALLBACK **GET_DLL_CAPS (**LPSTR lptr**)**

Upon start-up, Procomm Plus scans for any DL*x*s in the Procomm Plus installation directory. If one is found, Procomm Plus calls into the DL*x*'s **GET_DLL_CAPS()** entry point. **GET_DLL_CAPS()** sets bit flags in a "capabilities" word which is returned to Procomm Plus. The bit flag values are defined in **pwdefdlx.h** with the following meanings:

DLL_CAP_AUTODOWNLOAD (value 0x0001). The sender will transmit an auto-download sequence to the receiver. When Procomm Plus detects the sequence, it should automatically jump into file transfer mode. If the protocol supports auto-download sequences, Procomm Plus passes a pointer to a string (the **lptr** parameter) when it calls **GET_DLL_CAPS()**; the DLP should copy the auto-download target string to wherever the pointer is pointing. This string can be up to eleven characters in length, and may contain '?' wildcard characters indicating that any character will match at that character position within the string.

DLL_CAP_MULTISEND (value 0x0002). The protocol is capable of batching together several files and sending them with one file transfer request. Enabling this option allows a user to select multiple files in Procomm Plus's file transfer dialog box. You should also make sure bit **0x0004** = 1 to let Procomm Plus handle opening the files in the list.

DLL_CAP_PWOPENFILE (value 0x0004). Procomm Plus should open the file, as opposed to putting the file open code in the DLP. Normally set ON unless special file handling is required.

⇒ ***Note:*** *If this flag is not set ON, the DLP must open the file and update the same fields in* ***PROTOSET*** *in a manner similar to the Procomm Plus callback function,* ***P_FileRxTX****(). This flag should be set ON if multiple files are being sent as one batch. For more information about* ***P_FileRxTX()****, refer to the Procomm Plus Interface Specification.*

DLL_CAP_RCVNEEDNAME (value 0x0008). The protocol allows the filename of a transferred file to be included in the data stream, meaning that Procomm Plus doesn't need to prompt the user for a filename on the receiving side of the transfer. When the DLP receives the filename, it should call **P_FileRxTX()** to open the file.

DLL_CAP_SHOWXFER (value 0x0020). Procomm Plus should display the standard file transfer status box.

DLL_CAP_ADVANCED (value 0x0080). The protocol requires non-standard optional parameters, meaning that Procomm Plus needs to call the Advanced Setup dialog box. This call will take place when the user selects the Setup option from Procomm Plus's menu.

int CALLBACK **STARTDLL_XFER (**PROTOSET *pd, MAIN *mn, HWND hwnd,
struct ADVANCEDSET *AdvSet, int send, int from_auto**)**

Tells the DLP to initiate the file transfer. Procomm Plus passes in pointers to its ***pd*** and ***mn*** data structures, as well as two flags that tell the DLP whether the file is to be sent or received, and whether this request to start a transfer was the result of a received auto-download sequence.

One of the "capabilities" discussed in the previous section was a flag (**0x0004**) indicating whether the file would be opened by Procomm Plus or the DLP. If this flag = 0, the DLP must open the file. In this case, the **STARTDLL_XFER**() code is responsible for updating the **pd** data structure after the DLP opens the file. For more details, see the discussion of **P_FileRxTX**().

int CALLBACK **DLL_XFER (**PROTOSET *pd, MAIN *mn, HWND hwnd,
struct ADVANCEDSET *advset, int send**)**

Called periodically (approx. 18 times / sec.) by Procomm Plus. The sample DLP uses this pseudo timer interrupt to drive a rudimentary multi-tasking state-switcher while the file transfer is in progress.

Advanced Setup Entry Points

The remaining three function calls deal with the Advanced Setup Options Dialog Box. If your new protocol requires setting custom options, you may want to create a special Setup Options Dialog Box. If your protocol is selected as the current protocol, and the user selects the Procomm Plus Setup menu option, your dialog box will be displayed.

To create a custom Setup Options dialog, define your dialog box with ID number 14600; all controls on the dialog must be numbered in the range of 14601 up to 14699. Make certain that the DLP sets bit **0x0080** when the **GET_DLL_CAPS**() entry point is called. This will cause Procomm Plus to call your dialog box during Setup time. Also, use the 325 byte area pointed to by ***AdvSet** for saving any setup data, as Procomm Plus will save and restore that area.

void CALLBACK **INITDLGITEMS (**HWND hDlg, struct ADVANCEDSET *AdvSet**)**

Called by Procomm Plus prior to displaying the Advanced Setup Options Dialog Box. **InitDlgItems()** should initialize the dialog contents according to the protocol's default values.

BOOL CALLBACK **SETWM_COMMAND (**HWND hDLG,
struct ADVANCEDSET *AdvSet, int Id, int Cmd**)**

Procomm Plus inserts a hook into the Advanced Settings Dialog Box processing, allowing the DLP to perform any special processing when a related dialog control is selected or altered.

Id field refers to the ID of the control on the dialog box receiving this command (**Cmd**).

Cmd is a standard control notification code as defined in **windows.h.** For example, **CBN_SELCHANGE**.

BOOL CALLBACK **SETDEFAULTS (**HWND hDlg, struct ADVANCEDSET *AdvSet**)**

Called by Procomm Plus when the user selects "Use Defaults" from the "Modify Option Set" dialog accessed via the "Transfer Protocol" setup dialog. The various data fields should be reset to their default values.

BOOL CALLBACK **GETDLGITEMS (**HWND hDlg, struct ADVANCEDSET *AdvSet**)**

Called by Procomm Plus when the user closes the Advanced Settings Dialog Box. The dialog box's various field and checkbox values should be moved into their appropriate locations in the **AdvSet** structure.

Index of Functions

DLL_XFER 12

DllMain 11

GET_DLL_CAPS 11

GETDLGITEMS 13

INITDLGITEMS 13

P_DeletePartialFile 7

P_EndXfer 5

P_FeedDisplay 7

P_FileRxTX 4

P_LastError 6

P_ShowXferBox 5

P_StartGif 6

P_TimeToGo 6

P_XferFail 5

P_XferSuccess 5

SETDEFAULTS 13

SETWM_COMMAND 13

STARTDLL_XFER 12