

**Extending Procomm Plus®  
Using Dynamically Linked Connections**

<b>DLC OVERVIEW .....</b>	<b>3</b>
<b>NOTES ON THE SAMPLE FILES.....</b>	<b>4</b>
<b>DLL ENTRY AND EXIT.....</b>	<b>4</b>
<b>PROCOMM PLUS ENTRY POINTS.....</b>	<b>4</b>
<b>A Design Note .....</b>	<b>6</b>
<b>INTERNAL WORKER ROUTINES.....</b>	<b>14</b>
<b>INDEX OF FUNCTIONS.....</b>	<b>15</b>

## 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 DLx.

## DLC Overview

In order to support the wide variety of both present and future hardware and software connection types, Procomm Plus is designed to be flexible and extendable through the use of Windows Dynamic Linked Libraries (DLLs) specifically built to take advantage of these connections. These DLLs, renamed DLCs, allow Procomm Plus to redirect all normal Windows Comm function calls through the DLC itself.

The DLC contains function calls equivalent to those found in the Windows Comm engine. To allow Procomm Plus to store DLC setup information, and to communicate with the DLC, a pointer to 128 bytes (referenced below as **NETWORK** struct) is passed to the communication functions in the DLC. This block can be used by the DLC for any purpose desired.

If a connection DLL is detected by having an extension of **.dlc** and residing in the installation directory of Procomm Plus, it is loaded and queried for its capabilities, such as the name of its connection and whether it is a "version 4.00" compliant DLC. The DLC is given **Startup()**, **Reset()** and **Shutdown()** calls for its internal control; Procomm Plus expects the DLC to provide its own buffer space sufficient for its declared receive and transmit buffers.

⇒ **Hint:** *If the DLC needs to present a dialog, Procomm Plus passes in a window handle for this purpose. The DLC may also use the **MB\_TASKMODAL** flag in an independent dialog to prevent re-entrancy problems.*

⇒ **Hint:** *Any change in the **NETWORK** struct contents causes Procomm Plus to close and re-open the port or connection. This behavior can be used to display a dialog for a variety of purposes. For example, the **PWNCSI.DLC** uses this feature to display a port-selection dialog each time a port is opened or selected.*

The current state of the interface design is presented below. The following documents should also be considered part of the DLC specification:

- ◆ **bios.c** - Implements a simple connection using BIOS calls to the four standard Comm ports.
- ◆ **bios.h** - #include file containing structures and typedefs for the PW4BIOSDLC.
- ◆ **bios.def** - Linker module definition file for PW4BIOSDLC.
- ◆ **bios.rc** - Resource file for PW4BIOSDLC.
- ◆ **myint14.c** - Source to a worker function used by the main **bios.c** code.
- ◆ **makefile.dlc** - This is the makefile for building **pw4bios.dlc**.

There are three groups of functions within a DLC: the DLL entry and exit routine, the Procomm Plus entry points, and the internal worker routines.

## Notes on the sample files

The sample files included with this document produce a DLC called PW4BIOS.DLC. This DLC was originally written for 16-bit versions of Procomm Plus and has been ported to 32 bits. While the code should work if an int14 driver is installed, at least under Windows 95 and its successors, the ebios interface is generally outdated, and is probably not useful for production purposes. The sample files compile and integrate into Procomm Plus as described, but they should be viewed as samples rather than a product in and of themselves.

## DLL Entry and Exit

When a DLC (or any DLL) is loaded, Windows calls its **DllMain()** function.

⇒ *For more information, refer to the Windows DLL documentation.*

---

BOOL WINAPI **DllMain** (HANDLE hInstance, DWORD dwFunction, LPVOID lpNot)

⇒ **Note:** *We recommend that the DLC keep a global variable which is a copy of the **hInstance** handle for .DLL resources such as strings or dialogs.*

---

## Procomm Plus Entry Points

The second group of functions are called by Procomm Plus for a variety of purposes:

---

EXPORTED DWORD CALLBACK **C\_GetCaps** (void)

**C\_GetCaps** is a version checking function for Procomm Plus.

### **Arguments:**

None.

### **Returns:**

The DLC version value; currently, this should be at least 200. Since this documentation is concerned only with 32-bit versions of Procomm Plus, it is safe to use the value of 400 (indicating 4.xx). This value will change only when changes are made to the Procomm Plus specification.

---

EXPORTED DWORD CALLBACK **C\_AddNewConnection** (LPSTR line, int i)

**C\_AddNewConnection()** is called to build the list of available connections through the DLC. It is called when a user selects a port, or installs a new modem from the connection dialog.

**Arguments:**

**LPSTR line.** A pointer to a Procomm Plus string variable of 40 characters. The name of the connection is copied to this variable as it appears in Procomm Plus's *Setup* dialog port selection. For example, "Int14h - port1". If a connection is available and **C\_EnumConnections()** returns TRUE, this will contain the name of the connection.

**int i.** The number of connections currently reported to Procomm Plus by this DLC. Procomm Plus will continue to call **C\_EnumConnections()**, incrementing **i** until it returns FALSE.

**Returns:**

TRUE if a connection is available; FALSE otherwise. If TRUE is returned, the name of the available connection will be loaded into **LPSTR line**.

---

EXPORTED BOOL CALLBACK **C\_Setup** (HWND hWnd, LPSTR name, NETWORK \*nt )

**C\_Setup()** is called only when the user selects the **Modem/Connection Properties** button in the *Setup* dialog. Typically, **C\_Setup()** can be used to display an *About* box if no configuration is to be performed.

**Arguments:**

**HWND hWnd.** The currently active Procomm Plus window handle. This can be used as a parent window for dialog boxes.

**LPSTR name.** The full name of the connection as it appears in Procomm Plus's *Setup* dialog current connection. For example, "Intel 9600 Auto-Int14:1".

**NETWORK \*nt.** A pointer to a 128-byte block, which the DLC can use for any purpose.

**Returns:**

TRUE if setup was successful; FALSE otherwise.

**A Design Note**

When a Comm port is requested, Windows, opens the port and returns a handle. The handle is used as an argument to subsequent commands to reference the desired port. Procomm Plus's DLC format uses the identical functionality.

Because Windows is a multitasking environment, each DLC must allow multiple instances of Procomm Plus to speak to it. To that end, we recommend that any required variables be placed within a STRUCT, and that the STRUCT be globally allocated for each valid open connection. Typically, this would mean saving the handle to the STRUCT (or a pointer) in the NETWORK pointer passed by Procomm Plus with the **C\_OpenComm()** call. The NETWORK pointer will be passed into all subsequent Comm functions; the DLC may reference that pointer to access its data on a per-instance basis.

⇒ **Note:** *The sample DLC, **pw4bios.dlc** included with this document does not follow this example because of the limitations of the Int14h connection.*

---

```
int CALLBACK C_OpenComm ( HWND hWnd, LPSTR lptr, WORD txsize,  
                          WORD rxsize, NETWORK *nt, BYTE from_install)
```

Opens the specified port or connection, allocating the specified transmit and receive data buffers.

**Arguments:**

**HWND hWnd.** The currently active Procomm Plus window handle. This can be used as a parent window for dialog boxes.

**LPSTR lptr.** The name of the desired connection port as it appears in the *Selected Port* list box in the *Setup* dialog. For example, "Int14:1".

**WORD txsize.** The size of the transmit data buffer; Procomm Plus expects the DLC to maintain its own buffer space to satisfy the specified buffer size.

**WORD rxsize.** The size of the receive data buffer; Procomm Plus expects the DLC to maintain its own buffer space to satisfy the specified buffer size.

⇒ **Warning!** *It is important to maintain the **full** amount of buffer space specified in both the **rxsize** and **txsize** values. Some protocols, especially Zmodem, expect to be able to dump full buffer sizes of data to the port after validating available space!*

**NETWORK \*nt.** A pointer to a 128-byte block of memory, which can be referenced by the DLC for any purpose. Typically, this is used to implement multitasking considerations.

**BYTE from\_install.** Not currently used. Was previously used for compatibility with PROCOMM PLUS for Windows version 1.0.

**Returns:**

An integer, reflecting either the number of the open port, or an error condition. Error return values from **C\_OpenComm()** are **#defined** in **WINDOWS.H** as "IE\_XXXXXX". For example, **IE\_OPEN** and **IE\_MEMORY**.

⇒ **Note:** All return values from the DLC should function exactly as described in the standard Windows Comm functions.

---

int CALLBACK **C\_CloseComm** (int hCid, NETWORK \*nt)

Shuts down the specified port or connection, clearing and releasing any buffers allocated for that connection.

**Arguments:**

**int hCid.** The port or connection to be closed. This value should be the same value returned by **C\_OpenComm()** when the port or connection was opened.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**, above.

**Returns:**

TRUE if the close operation was successful; FALSE otherwise.

⇒ **Note:** When Procomm Plus attempts to close a port accessed by a DLC, it makes three different calls into the DLC. **C\_FlushComm()**, **C\_EscapeCommFunction()** and **C\_CloseComm()** are called. During these closing calls, the data contained in **NETWORK \*nt** will be invalid. If you need to reference specific data, use the Comm handle within your DLC.

---

int CALLBACK **C\_FlushComm** (int hCid, int rcv, NETWORK \*nt)

Clears any pending transmit or receive data from the specified port or connection queues.

**Arguments:**

**int hCid.** The port or connection whose buffers are to be cleared. This is the same value returned from the **C\_OpenComm()** call which opened the port or connection.

**int rcv.** If non-zero, indicates that the **receive** data buffer should be cleared. Otherwise, if zero, the **transmit** buffer will be cleared.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**, above.

**Returns:**

0 if the clear process was successful; 1 otherwise.

---

int CALLBACK **C\_ReadComm** (int hCid, LPSTR lptr, int cnt, NETWORK \*nt)

Reads characters from the specified port or connection.

**Arguments:**

**int hCid.** The port or connection whose received data buffer is to be read. This is the same value returned from the **C\_OpenComm()** call which opened the port or connection.

**LPSTR lptr.** A pointer to a string; if characters are successfully retrieved from the received data buffer, they will be loaded into the specified string.

**int cnt.** The maximum number of characters to retrieve from the received data buffer.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**, above.

**Returns:**

An integer reflecting the number of characters read from the received data buffer. If an error occurs in **ClearComm()**, this value will be negative.

---

int CALLBACK **C\_WriteComm** (int hCid, LPSTR lptr, int cnt, NETWORK \*nt)

Write characters to the specified transmit data buffer. **C\_WriteComm()** works exactly like the standard Windows function call **WriteComm()**, except that Procomm Plus passes in the **NETWORK \*nt** pointer.

**Arguments:**

**int hCid.** The port or connection whose transmit data buffer is to be fed. This is the same value as returned by **C\_OpenComm()** when the port or connection was opened.

**LPSTR lptr.** A pointer to a string containing the characters to be written to the transmit data buffer.

**int cnt.** The number of characters to be written to the specified transmit data buffer.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**, above.

**Returns:**

An integer reflecting the number of characters written to the transmit data buffer. If an error occurs in **ClearComm()**, this value will be negative.

---

int CALLBACK **C\_GetCommError** (int hCid, COMSTAT \*cstat, NETWORK \*nt)

Read and report status of the port or connection, returning a negative value in case of error. This function works identical to the standard Windows **GetCommError()** function.

**Arguments:**

**int hCid.** The port or connection whose status is to be tested. This is the same value as returned by **C\_OpenComm()** when the port or connection was opened.

**COMSTAT \*cstat.** A pointer to a data structure whose fields provide port or connection information. This structure is declared in **windows.h**.

⇒ **Note:** For more information on the **COMSTAT** struct, refer to the Windows SDK documentation.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

1 if an error condition is reported; 0 otherwise.

---

int CALLBACK **C\_UngetCommChar** (int hCid, char c, NETWORK \*nt)

Adds a character to the front of the received data buffer, just as if it were received normally, and the next character to be read.

⇒ **Note:** Procomm Plus 4.xx will **never** call this function; it has been eliminated from the Procomm Plus specification.

**Arguments:**

**int hCid.** The port or connection whose received data buffer is to receive the “ungotten” character. This is the same value as returned by **C\_OpenComm()** when the port or connection was opened.

**char c.** The character to be written to the port or connection’s received data buffer.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

TRUE if the character was written to the buffer; FALSE otherwise.

---

int CALLBACK **C\_SetCommBreak** (int hCid, NETWORK \*nt)

Initiates a BREAK signal. This function is identical to the standard Windows API function, **SetCommBreak()**.

**Arguments:**

**int hCid.** The port or connection on which the BREAK will be generated. This is the same value as returned by **C\_OpenComm()** when the port or connection was opened.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

TRUE if the BREAK was generated; FALSE otherwise. Also returns TRUE if a BREAK is not supported on the port or connection

---

int CALLBACK **C\_ClearCommBreak**(int hCid, NETWORK \*nt)

Terminates a BREAK signal. This function is identical to the standard Windows API function, **SetCommBreak()**.

**Arguments:**

**int hCid.** The port or connection on which the BREAK will be terminated. This is the same value as returned by **C\_OpenComm()** when the port or connection was opened.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

TRUE if the BREAK was terminated; FALSE otherwise. Also returns TRUE if a BREAK is not supported on the port or connection

---

int CALLBACK **C\_SetCommState**(DCB \*dcbw, NETWORK \*nt)

Sets the port or connection line characteristics. For example, the baud, parity and databits. This function is called by Procomm Plus to set the port or connection state, and is identical to the standard Windows API **SetCommState()**.

**Arguments:**

**DCB \*dcbw.** A pointer to a Device Control Block structure, which is **#defined** in **windows.h**.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

TRUE if the device was successfully set to the specified parameters; FALSE otherwise.

---

int CALLBACK **C\_GetCommState** (int hCid, DCB \*dcbw, NETWORK \*nt)

Reports the port or connection line characteristics. For example, the baud, parity and databit settings. This function is called by Procomm Plus to retrieve the port or connection state, and is identical to the standard Windows API **GetCommState()**.

**Arguments:**

**int hCid.** The port or connection whose status is to be read. This is the same value as returned by **C\_OpenComm()** when the port or connection was opened.

**DCB \*dcbw.** A pointer to a Device Control Block structure, which is **#defined** in **windows.h**.

⇒ **Note:** For more information on the DCB struct, refer to the Windows SDK documentation.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

TRUE if the device was successfully set to the specified parameters; FALSE otherwise.

---

int CALLBACK **C\_EscapeCommFunction** (int hCid, int action, NETWORK \*nt)

Sets the DTR line or simulates a received XON character. This function is identical to the standard Windows API **EscapeCommFunction()**.

**Arguments:**

**int hCid.** The port or connection whose status is to be read. This is the same value as returned by **C\_OpenComm()** when the port or connection was opened.

**int action.** The action to be performed. Refer to the Windows API documentation for acceptable values.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

TRUE if the specified action was successfully performed; FALSE otherwise.

---

int CALLBACK **C\_DCD** (unsigned port, NETWORK \*nt)

Returns the status of the DCD line (Carrier Detect).

**Arguments:**

**unsigned port.** The port or connection DCD line is to be tested. This is the same value as returned by **C\_FindPortAddress()**.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

1 if Carrier Detect is high; 0 if low.

---

int CALLBACK **C\_QueryBaudRate** (int baud, NETWORK \*nt )

Called by Procomm Plus to determine whether a baud rate is supported by the DLC.

**Arguments:**

**int baud.** The baud rate to be tested. Refer to **windows.h** for acceptable values.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

If the DLC cannot support the requested baud rate, it returns the highest supported baud rate. Otherwise, it returns the requested baud rate.

---

int CALLBACK **C\_FastRate** (int baud, unsigned port, NETWORK \*nt)

Sets the port or connection baud rate if greater than 19200. This function was used for Windows 3.0, and is no longer needed for Windows 3.1 and beyond. However, it must be present in the DLC.

**Arguments:**

**int baud.** The baud rate to be set. Acceptable values are 7 (38400), 8 (57600) and 9 (115200).

**unsigned port.** The port or connection whose baud rate is to be set. This is the same value as returned by **C\_FindPortAddress()**.

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

TRUE if successful; FALSE otherwise.

---

unsigned CALLBACK **C\_FindPortAddress** (NETWORK \*nt)

Called by Procomm Plus after a port or connection is opened to determine its address. This function was used for Windows 3.0, and is no longer needed for Windows 3.1 and beyond. However, it must be present in the DLC.

**Arguments:**

**NETWORK \*nt.** A pointer to a 128-byte block of memory, as described under **C\_OpenComm()**.

**Returns:**

The port or connection address, or **0** if this address is not required by the DLC.

## Internal worker routines

Internal worker routines will be DLC-specific; refer to the source code for **pw4bios.dlc** for examples.

## Index of Functions

<b>C_AddNewConnection</b> .....	<b>5</b>
<b>C_ClearCommBreak</b> .....	<b>10</b>
<b>C_CloseComm</b> .....	<b>7</b>
<b>C_DCD</b> .....	<b>12</b>
<b>C_EscapeComm</b> .....	<b>11</b>
<b>C_FastRate</b> .....	<b>12</b>
<b>C_FindPortAddress</b> .....	<b>13</b>
<b>C_FlushComm</b> .....	<b>7</b>
<b>C_GetCaps</b> .....	<b>4</b>
<b>C_GetCommError</b> .....	<b>9</b>
<b>C_GetCommState</b> .....	<b>11</b>
<b>C_OpenComm</b> .....	<b>6</b>
<b>C_QueryBaudRate</b> .....	<b>12</b>
<b>C_ReadComm</b> .....	<b>8</b>
<b>C_SetCommBreak</b> .....	<b>10</b>
<b>C_SetCommState</b> .....	<b>10</b>
<b>C_Setup</b> .....	<b>5</b>
<b>C_UngetCommChar</b> .....	<b>9</b>
<b>C_WriteComm</b> .....	<b>8</b>