



Welcome to CommX 1.10

Release Notes

We hope your experiences with Greenleaf CommX 1.1 are as enjoyable as ours were bringing the product to you. Please keep in touch via email and check our Web site often for added Examples, FAQ items, and other product news!

Please take a few moments to read through the Release Notes – it could save you some time and effort. Some things are new, some changed, some fixed.

The CommX Team
Greenleaf Software Inc.
<http://www.greenleafsoft.com>
info@gleaf.com



Release Notes: CommX 1.10

Release Notes

General: Release Notes and Advisories

- As you use CommX, you should check the FAQ on our Web site, <http://www.greenleafsoft.com> often as we post new and significant information there often. In addition, as the *Greenleaf Chronicles* newsletter is published, it will contain articles of interest for users.

Important Topics in Release Notes

- CommX Examples & Tutorials
- Event Mode – new PortCtl Property (added in version 1.1)
- ZModem Notes – Some changes
- Registry: Visual C++ and Redundant Registry Entries
- Sybase Power++ 2.1 User Advisory
- RxTriggerChar / IncomingFlagChar Usage Notes
- CommX 1.1 Bug Fixes and Compiler Update Notes
- Delphi 4 – Important Information
- Delphi 3 – Important Information
- PowerBuilder 6 Notes
- Visual FoxPro 5 Notes
- Avoid Timed Methods in Event Handlers
- Note about Close Method
- Working With a Modem



Note About Close Method

Release Notes

Implicit Close (Port)

It is not necessary to call the **Close()** method explicitly in all cases. When the application terminates, the **PortCtl** object is destroyed and the port conditions (stored in DCB and other serial comm structures) are restored to their state prior to creating / opening the port by CommX.

- However, and especially if you are using Visual C++ 5.0 and above and you are using a **TermCtl** and/or **FileXferCtl** object, you must call the following methods prior to exiting your application:

```
FileXferCtl.SetPort(0);  
TermCtl.SetReadingPort(0);  
TermCtl.SetReadingKeyboard(0);
```

- The reason for these calls is that the pointers that establish the relationship between the **PortCtl** and other CommX objects need to be deleted prior to exit.



Working With a Modem

Release Notes

How to Dial a Modem

You can use the `WriteString()` method to dial a modem. For example:

```
MyPort.WriteString( modemCommandString );
```

Where *modemCommandString* is a telephone number preceded by the modem command to dial, e.g. "ATDT," and followed by a Carriage Return (decimal 13).

For example, to dial the number 800-555-1212, the code might be (Visual Basic):

```
MyPort.WriteString ("ATDT1-800-555-1212" & vbCr)
```

Exit From Online Mode

- After your modem enters data (online) mode, in order to return to command mode, you will need to send a string consisting of about 3 of the *Escape Code* values as stored in the modem's S2 register. The default and most common value for the *Escape Code* is '+' (decimal 43). Sending the string "+++" is the most common way to "escape" from online mode and enter command mode (at which time the modem will respond to further commands).
- After you send the Escape Code ("+++") as above, there is a delay that is invoked by the AT command compatible modem. The delay is in units of 1/50th second and is stored in the modem's S12 register. The default and commonly used value is 50 (i.e. 1 second). The modem waits this amount of time for anything that is going to occur on modem status or data lines to settle down before changing the modem to command mode. In command mode, it is insensitive to data from the phone line.
- In order to successfully make the transition from online mode to command mode, code such as the following is suggested (Visual Basic example):

```
MyPort.WriteString( "+++" )  
Sleep(2000)           ' Or other means of delaying here  
PortCtl.WriteString( "          vbCr )      ' Command to go on-hook (hang up)
```

- For further information (a complete listing of modem commands) see your modem instruction manual or the Appendix topic *Modem Command Reference*.



Using Timed I/O Methods

Release Notes

General

The timed methods for reading and writing data to the port will hold up processing in the calling thread of execution while waiting for the operation to complete. If this is not desired, consider creating your own timeout period and call the corresponding methods that do not wait, calling them repeatedly until your timeout period has elapsed.

Discussion

Calling **ReadStringTimed** or **ReadByteBufferTimed** in the **IncomingData** event handler or in the **IncomingFlagChar** event handler can also cause unwanted delays in your code.

The timed reads will always time out if the expected data has not already been read from the port when the event was fired.

IncomingData Event Handler Considerations

The **IncomingData** event is fired when data has been received from the port, and additional data will not be read from the port until the **IncomingData** event handler has returned control to CommX. Placing a timed read method in the **IncomingData** handler will cause a delay of the specified number of milliseconds if the expected data has not already been read from the port by the time the event was fired.

IncomingFlagChar Event Handler Considerations

The **IncomingFlagChar** event is fired when the expected flag character has been read from the port. If additional characters are expected and haven't been read from the port at the time this event was fired, the timed read method used in this event handler will cause a delay of the specified number of milliseconds.

Timed Write Methods

Using the timed write methods (**WriteStringTimed**, **WriteByteTimed**, **WriteByteBufferTimed**) can also cause unexpected delays by holding up processing of the calling thread of execution while waiting for the operation to complete. The timed write methods can still complete successfully in the **IncomingData** and **IncomingFlagChar** event handlers, but can cause unexpected delays depending upon the timeout value and the amount of data already in the transmit buffer.



Examples

Release Notes

Examples

We have posted a number of examples and some Windows Help based tutorials relating to getting started with a variety of development environments on our Web site at <http://www.greenleafsoft.com>. These will ship concurrent with this (1.10) version of the software. Included are examples for Visual Basic 5 and 6, Visual C++ 5 and 6, Visual FoxPro 5, Delphi 3 and 4, Power++ 2.1, PowerBuilder, and others. We are continuing to develop a knowledge base on using the software with an expanding base of environments for Win32.

All examples have been tested in Windows 95, 98, and NT 4.

- Remember: We are shipping *project files* which you need to load into your development environment and, should you choose, build into executable form. Though we *do* also ship executables, we **do not recommend** that you attempt to run them. The reason for this lies in the dependencies—the files, DLLs and other Microsoft or other development system redistributable files that are *required* for the executable to run. For the simple reason that you may be using a different version of Windows, updated or different DLLs, we cannot state that the executable we built in a particular environment will run as compiled.
- We do **not** ship the examples as packaged products; were we to do so we would be shipping redistributable DLLs and other files from a variety of sources—and these have a tendency to become obsolete about an hour after we ship our product. Therefore, please consider the examples for all environments as projects, not products.



PowerBuilder 6 Notes

Release Notes

Visibility at Design Time

The **TermCtl** object in CommX 1.0 was not visible at design time. This has been corrected; Terminal Control objects are now visible at design and run time.



Visual FoxPro Notes

Release Notes

Visibility at Design Time

The **TermCtl** object in CommX 1.0 was not visible at design time. This has been corrected; Terminal Control objects are now visible at design and run time.

Getting Events to Work

When using CommX with Visual FoxPro 5 or 6, you should set

```
Application.AutoYield = .F.
```

in your **Form.Load** procedure or any time before your application will begin handling events. It is also prudent to set

```
Application.AutoYield = .F.
```

in the **Form.Unload** procedure. VFP will handle events properly given this consideration.



Event Mode Description

Release Notes

New EventMode Property for Port Control

We discovered that a new property was required to properly facilitate event firing across thread boundaries in certain environments only. Since the CommX port software uses multiple threads, it was possible for events to be missed under some conditions.

- **EventMode** provides for proper marshalling to facilitate cross-thread event firing. This means that your primary thread can count on receiving events fired by other threads.
- The new property, **EventMode**, is especially important to Visual Basic 5 users **only when you step-debug through an event handler**.
- **Visual FoxPro applications** also need to have **EventMode** set to 1.
- **All other projects** should leave **EventMode** set to the default (0).
- See the CommX PortCtl documentation page for additional information about the use of **EventMode**.



Zmodem Notes

Release Notes

ZModem – Fixed / Improved

- Events were being fired even when they were disabled – changes described below prevent this and ZModem transfers are now more reliable.
- If you have difficulty getting ZModem to reliably transfer files, try increasing the size of the sender's sliding window to 4096. If too many errors are still evident, try disabling some of the status events for the PortCtl object. Properties you may wish to set to 0 (disabling the events) include **IssueCtsChangeEvent**, **IssueDsrChangeEvent**, and **IssueCdChangeEvent**. When you are transferring files with ZModem at high baud rates, the overhead and chance of error are increased by forcing these events to occur. Especially when using RTS/CTS handshaking (almost mandatory for ZModem), the **CtsChange** event is counterproductive and should *not* be used. The same applies to the **DsrChange** event, which is definitely counterproductive if using DSR/DTR handshaking, and probably not helpful otherwise.



Explanation

- Although the implementation of ZModem contained in CommX 1.0 is derived from some of the most experienced and reliable ZModem protocol software around (Greenleaf CommLib and Comm++), we have made changes which we believe will solve some file transfer operations.
- When modem status is being read from the port often (as is the case when RTS/CTS or DTR/DSR handshaking is employed with ZModem), the chances of misreading modem status interrupts is significant when using a small receive buffer. Essentially, we have prevented CommX from reading the modem status unless a modem status interrupt occurs, which greatly the probability of error.



Note about Visual C++ and the Windows Registry

When you unregister (e.g. REGSVR32 -U COMMX.OCX) ATL based ActiveX controls built with Visual C++ 5, some CLSID registry keys are not removed. The article below shows how to fix this and also states that the problem is fixed in Visual C++ 6. We have made the indicated modifications to the CommX code to avoid this problem. CommX 1.0 users should notice that this problem is not present with Version 1.1

Microsoft MSDN KnowledgeBase article ID Q186391 can be found at <http://support.microsoft.com/support/kb/articles/q186/3/91.asp>

Another KB article, Q179688 relates to the same basic problem.

Both articles are FYI and not required reading for using CommX.

CXClean Utility

With this release a new utility CXClean.exe will be run quietly during Setup. This will search the Windows registry for existing entries for CommX and delete them. This avoids multiple entries as noted above.

If you have multiple installations of CommX.ocx, you can use CXClean to destroy existing CommX 1.0 entries on each system.

If you get LoadLibrary Failed error when using REGSVR32.EXE

If this error occurs, check to be certain that you have copied COMMX.OCX to your \windows\system (Windows 95/98) or \windows\system32 (for NT) subdirectory, i.e. the same directory where REGSRVR32.EXE is located OR that your PATH includes the directory where REGSVR32.EXE is located.

Find the utility REGCLEAN.EXE. If you can't locate this, you can download it from the Microsoft web site or contact Greenleaf and we'll email it to you. Run this utility to successfully register CommX. On the Microsoft site, this file is at <http://support.microsoft.com/support/kb/articles/q147/7/69.asp> – or search the Knowledge Base for Article ID Q147769. The current version of this utility at the time this documentation went to press is 4.1a, build 7364.1 – 800,136 bytes.



Using Power++ 2.1

Release Notes

Sybase Power++ 2.1 Users:

When you import **any** dual mode ATL based ActiveX component into this development environment, you will see an icon on the Component Palette, ActiveX tab for the control **and also one for each property page** (if any) in the control. This is not specific to CommX. In the case of CommX, you will notice 8 spurious Leaf icons on the palette. If you mouse over any of these you will notice that they are like PortProp, PortProp2, TermProp, etc.

- You can safely ignore these spurious icons. We have notified Sybase of this issue and anticipate a solution soon.
- In the meantime, we are aware of no other anomalies regarding use of CommX in Power++ 2.1.
- In particular, do **not** try to place these property pages independently or outside the context of the actual Port, FileXfer, and Terminal Controls. The property pages must be accessed by right-clicking the control on your dialog or form or by calling the appropriate **Configure** method.
- EventMode provides for proper marshalling to facilitate cross-thread event firing. This means that your primary thread can count on receiving events fired by other threads.



Rx Trigger Character Events

Release Notes

About Rx Trigger Character Events in PortCtl

There are two mechanisms in CommX which allow client software to respond to preset character values to trigger an event:

1. Set the Port Control **RxTriggerChar** property and write a handler for the **IncomingFlagChar** event.
2. Establish a Terminal Control trigger string (which can be 1 or more characters in length) by using methods like **AddTriggerString** and writing a handler for the **IncomingTrigger** event.

We believe the most useful for most applications would be the Terminal Control event. However, you're the application developer so you get to choose. Here are some notes about the Port Control method

Incoming triggers are quite useful; however we have noted that some users were expecting characters in addition to the flag character to be present in the Rx Buffer when the **IncomingFlagChar** event occurs. Given that a flag character is sent after a group of characters, it will be received after the other characters. Last out, last in.

- The **IncomingFlagChar** event is fired to notify you that the flag character has been seen, and *not* that any additional information remains in the Rx Buffer.
- Any Read operations that you do prior to receiving the event may have removed all or part of the additional characters.
- If you **expect the flag to be received last** and want a contiguous block of data including / associated with the flag to be removed en masse, wait until the event occurs, then do the read operations in the event handler. You could use a loop like **While SpaceUsedInRxBuffer > 0** to control this transfer.
- If the **flag** is expected to be received first (or in the middle) in a block, you will need to plan I/O on the Rx Buffer accordingly.



Bugs Fixed / Compiler Updates

Release Notes

Unreported Bugs Fixed:

We are embarrassed to report that several bugs were present in the low level CommX software that were not detected by us or by Visual C++ 5. These have been fixed. The area involved was related to the number of stop bits forced when you set the data bits property to 5 (or set stop bits = 1.5, which should force 5 data bits). Fortunately, Visual C++ 6 helped us discover the cause of these and they have been repaired in CommX 1.1.

Documentation Changes

The Object Pascal compiler in Delphi 3 allowed us to do something that is fundamentally wrong when passing a reference to a byte array to a method like WriteByteBuffer:

```
Var  
  buffer[100 : Byte  
begin  
  object.method( buffer[1], ... );
```

This code worked but is incorrect Pascal and Delphi 4 discovered it for us. The correct reference is:

```
object.method( buffer, ... );
```



Delphi 4 Notes

Release Notes

If you are a Delphi 4 user...

July 14, 1998

If you are using Delphi 4, Build 5.37 (the initial release of Delphi 4), you will need to contact Technical Support at Inprise – <http://www.inprise.com> – for an update that enables dual-mode ATL based ActiveX control support. This bug has been confirmed by Inprise and a solution is available.

- Without this fix, you will see multiple errors when using CommX controls in your projects.



Delphi 3 Notes

Release Notes

If you are a Delphi 3 user...

April 12, 1998

We have determined that some versions of Delphi 3 Professional **do not permit use** of Greenleaf CommX. If you have one of the builds for which CommX does not work, please contact Borland International (now called Inprise Corp)

Specifically, we know that Build 5.53 does not, and that 5.83 works great.

- Borland has informed us that the correct version as of this writing is 3.01 plus downloadable Delphi component updates from www.borland.com (now www.inprise.com)



Redistributable Files

Your Products

Greenleaf CommX is intended to be incorporated in products which you develop and distribute or ship internally or to others. The files from CommX which you will need to distribute are listed below. These should be installed when your Setup utility sets up end user computers with CommX enhanced products.

File	Purpose
CommX.ocx	The CommX ActiveX controls
CommX.fon	Terminal Control (bitmap) fonts
CXClean.exe	Utility for removing old CommX registry entries
PortCtlHelp.hlp	Help for the Port Control property pages
TermCtlHelp.hlp	Help for the Terminal Control property pages
FileXferCtlHelp.hlp	Help for the File Transfer Control property pages

- If you do not use the Terminal Control in your applications, you can skip CommX.fon.
- The three property page group WinHelp 4 files are optional, depending upon your needs.
- The CommX documentation is, of course, not redistributable.

