Programming Serial Ports and Multi Port Devices with wxWindows and CTB

by Jürgen Schuhmacher

Version 0.11

August 14th, 2002

Topics

About this Document

Intention

This document deals with the steps you have to do (and the possible problems to solve) to get a multiport serial device (AKA port server) run on a Windows System and write application programs with wxWindows and the additional CTB-Library for serial programming. I will cover all the aspects I came across when getting started with my personal work. Due to certain demands, I decided, to do the cross platform development with wxWindows in my case rather than using Win32/MFC and porting it later to Linux/BSD.

Allthough I want to concetrate on wxWindows in the first place, platform specific aspects are covered though – see the topics table for details. I hereby do refer to Windows programming and especially to Microsoft's "perfect" OS WinXP (sometimes W2000) at the moment, but a Linux-specific part will hopefully follow. Some hints are yet included in the text.

Beginners might find several hints and tips to get over problems typically occuring, when first dealing with wxWindows and serial programming in general, since of course you won't need a multiport device to do simple serial programming. ☺ You even do not need wxWindows to start with serial programming – but those, who really want to jump into object oriented programming are encouraged to have a look at this fine environment acting as a wrapper library and RAD-tool as well.

Acknowlegements

WxWindows

wxWindows is a free library for cross platform GUI development, invented by Robert Röbling, Julian Smart and Vadim Zetlin. See http://www.wxwindows.org for details.

CTB-Library

CTB is a free library for writing cross platform applications requiring serial support, provided by Joachim Buermann. See http://www.iftools.de/ .

Multiport

Some information is taken from the documentation of my multiport device. I do not want to advertise for this particular company – so I tried to keep all information abstract.

Enjoy !

Dipl.-Ing. Jürgen Schuhmacher
jschuhmacher@gmx.de

Legal issues

Warranty

This document is NOT part of the official wxWindows documentaion. It just shows what I did to get my programs run and should be interpreted as a little helper for beginners. I cannot grant that things shown here are perfectly right or "the correct way to do" – also NO warranty is taken for any harm occuring when making use of the advice given in this paper !

Copying

Improvements

Where to get it

The newest version of the document as well as sample code can be found on my website. The Code mentioned in this document should be bundled with the paper. Reading the document, you will find new sections in comparison to the former version in blue colour. This might help to find new information quickly. Very important new information like error corrections are kept in red !

What to do :

Well, my todolist is still large, I know – but this document has just started.

to do in the short run :

Code of Win32 Samples
Overview "Programming issues"
wxWin Samples
Installation Issues Win 2000

and in the long run :

Full Part "Programming issues"
Installation Issues Win 2000
Installation Issues Linux
Programming Issues Linux / BSD
Porting Multiport Samples to Linux

Installation of WxWindows

Installing Visual Studio 6.0

After upgrading to WinXP and starting to look for a programming environment, I decided to keep the existing Visual Studio 6.0a Environment as the IDE für wxWindows rather than stepping to Visual Studio.NET. Currently VS6.0 is avalable as an “advanced” version 6.0a.

Installation on XP

Installing Visual Studio 6.0a did not work directly on my Office PC. The PC runs with Win XP Pro and refused to install the VS 6.0. due to an error in the setup routine. There was no problem when installing it on a Win2000-System however.

I had to install JAVA from the bundled JAVA-CD first to succeed. There is a note on MS Website concerning VS 6.0 on XP !

Environment Variables

After installing VS6.0, you will find a VSVARS32.BAT file in your installation. This is required for working with the vs tools (compiler, linker) from the command line. See the details in the Visual Studio Doc. I did not make use of it however.

Installing the wxWindows Library

I installed the library into directory C:\wx2.

The Environment Variable

In order to make sure, the WXWIN environment variable is set correctly, you may add the tag “SET WXWIN = C:\WX2" to the autoexec.bat.

Win XP, you might set the variable from within the system control box :

[START][Settings][System][System Properties]

[ADVANCED] … in box 3 : [Environment Variables]

add name : “WXWIN” and value “C:\wx2".

Creating The Libraries

General Meaning

To use wxWIn, you may want to create certain different libraries such as for Release / Debug and DLL / non-DLL. The meaning od this :

When creating programs later, you have the opportunity to provide a stand alone file having all necessary functions included - or much smalle file requiring the appropriate wxWin-Lib (such as e.g. wx22_9.lib) to run. For both of the program versions, you can create the final release version as well as a debug version for the developing process.

How To Build The Libs

From the MS Visual Studio IDE, open one of the two given workspaces "wxvc.dsw" or " "wxvc_dll.dsw", that can be found in the sources directory. In [Build][Set Active Config] choose "Debug" or "Release". Selecting all possible configurations, you might finally do runs for :

a) wxvc_dll – Debug
b) wxvc_dll – Release
c) wxvc – Debug
d) wxvc – Release

and eventually also Debug / Release version for "jpeg", "png", "tiff", "xpm" and "zlib". Your directory "lib" under "wx2" might finally look like this :

```
10.06.2002  16:49    <DIR>          bcc16
10.06.2002  16:49    <DIR>          watcom
20.05.1998  14:02                57 dummy
10.12.1999  12:45               330 vms.opt
07.03.2000  09:00               364 vms_gtk.opt
20.07.2000  04:09           600.617 wx22.def
28.06.2002  14:18         2.490.368 wx22_9.dll
28.06.2002  14:17         3.614.742 wx22_9.lib
28.06.2002  14:17         2.232.538 wx22_9.exp
28.06.2002  14:31        27.670.244 wx22_9d.ilk
28.06.2002  14:31         4.792.359 wx22_9d.dll
28.06.2002  14:30         3.644.186 wx22_9d.lib
28.06.2002  14:30         2.246.471 wx22_9d.exp
28.06.2002  14:31         7.422.976 wx22_9d.pdb
28.06.2002  14:38           199.014 jpeg.lib
28.06.2002  14:42           765.644 jpegd.lib
28.06.2002  14:38           201.948 png.lib
28.06.2002  14:42           350.646 pngd.lib
28.06.2002  14:39           437.494 tiff.lib
28.06.2002  14:43           845.954 tiffd.lib
28.06.2002  14:37            71.944 zlib.lib
28.06.2002  14:37           126.482 xpm.lib
28.06.2002  14:41        10.307.558 wx.lib
28.06.2002  14:41           141.594 zlibd.lib
28.06.2002  14:42           299.032 xpmd.lib
28.06.2002  14:44        20.707.126 wxd.lib
```

… where "d" indicates the debug version if a library. Now, you are ready to run the samples coming with the wxWin-Lib.

The wxWin Samples

Unlike many other distributions of libraries, wxWindows comes with a number of usefull samples which not only show the capabilities of this Lib, but could also act as a good starting point of own applications. (See below).

Building the Samples

Open the general workspace for VS covering most of the samples to import all the projects at one time. If there is no DSW/DSP for some projects , select “cancel” to skip them.

Select the “Release”-Config in [Build][Set Active Config] to obtain release versions, and do a “build”. The executables will appear in the sub directories named “Release”.

Usefull Sample Programs

The IMHO most interesting samples are for instance the (dynamic) menu demonstration and the tool bar samples. Generally you will find most of the required functions of your application explained demonstrated some where in the samples. Beginners should start with the minimal sample to get an idea how to start with wxWin.

Creating A New Project

An easy way to start with an own application is to use one of the samples and copy it (with a new name) into the samples directory. Thus you can use all environment and path settings, which might be confusing for beginners. You should try to get familiar with them though.

Serial Communications in Windows

In this chapter you will find some hints on general serial programm for win32 and multi port server. The special aspects of wxWindows and the the CTB-Library are covered later.

Programming Techniques

Serial Programming Basics

To follow

Accessing COM ports

In Windows, COM ports are accessed by demanding the devices "COM1, COM2". According to your specific installation, you eventually will find more than the two common ports, most likely named "COM3, COM4". In Linux you will find "`/dev/cua0" , "/dev/cua1" and so on.`

Reassigning Port Names

The port names themselves are thereby memorized in the Windows Registry. Currently I cannot make suggestions how and if it is possible to easily change them in the different Windows version . However, so the annotations in the multi port chapter.

Information is welcomed. ☺

More Ports Than Usual

Windows (XP / 2000) can deal with up to 1024 ports at a time. Adding more IO-cards or port servers will automatically create more port names like shown above. But Windows has problems in accessing port numbers the known way, when the number is greater than 9 ! So you will have to use the general string "\\.\COMx" instead. When using a device name such as "COM24" you will have to use "\\.\COM24" instead in order to get access to this particular port.

A Programming Problem

A further tricky thing, where beginners might stumble over, is the "\"-issue in sting constants. When preparing a string with the printf-function, the backslash acts as an indicator for a special character to follow and is thus not displayed as a selfstanding character. You must use two slashes ("\\") to finally obtain exactly one slash in the string - so the full phrase for a port name will be : "\\\\.\\COMx" A complete code snipplet for win32 could look like this :

```
hcom = CreateFile(  "\\\\.\\COM1",
                    GENERIC_READ | GENERIC_WRITE,
                    0,
                    NULL,
                    OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL,
                    NULL
                   );
```

Parts of a Program

Settings

To follow …

Opening a Port

To follow …

```
hcom = CreateFile(  "portname",
                    GENERIC_READ | GENERIC_WRITE,
                    0,
                    NULL,
                    OPEN_EXISTING,
                    FILE_ATTRIBUTE_NORMAL,
                    NULL
                    );
```

Read Operation

To follow …

Write Operation

To follow …

Multiport Servers

Once again, you do not need a multiport device in any way. All discussed aspects here refer to (and directly work with) the known physical ports in your PC. However, there are some special aspects when working with the multi port device drivers. I am discussing them to give you an idea about how to deal with multiport devices in particular.

Function and Benefits

With the help of a multi port server or an multi port io-card, it is possible to accesss more than two serial devices with a common PC. Usual systems come as PCI cards offering e.g. 2/4/8 additional COM ports of mostly RS-232 type, but some of them do also provide ports according to the RS-422 and even RS-485 specification.

The port server, focussed in this particular case, can be connected directly to the NIC or also over the LAN to an e.g. Server-PC. In the second case, the port server need not be located in beneath of the PC. The device driver installed on the Server PC, offers the virtual serial ports to any application. A maximum speed of 230kB/s is available for all of the 16 Ports per unit, where more than one device could be used at a time.

The adressing works via MAC-/ or IP-adressing. The port server is configurable from within its device driver (windows control panel) or also with a website – even over an internet connection. The multi port devices come with signal latencies around (min/typ/max) 7/15/20 ms, where the different versions of Windows "offers" up to 100ms in some cases. Drivers are available for most operating systems – mainly for server systems like Linux / W2000.

Installation Configuration

Setting new Port Names

The multiport device driver used here offers a number of changeable alias names for the individual ports. When installing the 16 channel device, the ports COM3 ... COM18 do appear in the windows control panel ([SYSTEM][DECICES] and hopefully also in the listboxes of the applications. It is thus possible, to select the desired COM port to work with. Sometimes you will have to specify the port explicitely e.g. in a configuration file - if the application does not gather the port names from the registery. However Windows Hyperterminal does recognize the new portnames after reboot.

Installing more than one multiport device will give you more of the above port names, such as COM19, COM20 and following, according to the installation sequence of the devices. Since this might cause confusion, it is usefull to group the ports of one device. For a better handling, I renamed the ports to D01P01 ... D01P16, and D02P01 ... D02P16 to distinguish devices and ports easily. Here, this has to be done from within the multiport's device manager, which also allows to determine watchdog settings and timeout parameters.

Programming Issues

Opening A Port

Regarding the paragraph about how to assign ports, a special naming has to be applied, when opening a file. When using a device name such as "D02P05" (assigned in our multiport's device manager box) you will have to use "\\.\D02P05" to access to this port.

Code might look like this : (DEV-CPP)

```
strcpy(namestring, "\\\\.\\");
strcat(namestring, comname);
port = CreateFile(namestring,
            GENERIC_READ | GENERIC_WRITE,
            0,                                          // no sharing
            NULL,                                       // no security
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED,
            NULL);

if (port == INVALID_HANDLE_VALUE)
{
            Printf ("OpenPort Fail %s\n", comname);
            status = GetLastError();
            Printf("Error from CreateFile: %d",status);
   return -1;
}
```

Refering to our multi port devices, the port #12 of the device #2 will then be named "\\\\.\\D02P12" - see the full source code for details !

Buffer Settings

According to the manufacturer's advice, the write buffer should be as large as possible - up to a maximum of 1400 kB. The read buffer size does not matter. I observed no change in the behaviour of the bevice, experimenting with the buffer.

Program Settings

The first issue is to simply regard the latencies and always make sure, not to exceed the connections cababilitiy. There might be security problems when using LAN instead of a point-to-point connection between multi port device and the NIC. Data might be corrupted or missed easily, which should lead to a more careful data handling. A special logical protocol including repeated checking and resending data might be suitable.

Timing Parameters

The multiport device does not completely behave like the driver of the physical devices of courdse. Allthough considered to be fully compatible, I observed some stops and unwanted delays when reading a byte using "my" way of timing, I allways used with physical ports.

Reasons for this could be the permanent latencies of windows and sometimes delayed data transaction due to data traffic in the LAN. According to the win32 manual, I found the following working configuration for my win32 application.

```
tout.ReadIntervalTimeout = MAXDWORD;
tout.ReadTotalTimeoutMultiplier = 0;
tout.ReadTotalTimeoutConstant = 0;
tout.WriteTotalTimeoutMultiplier = 1;
tout.WriteTotalTimeoutConstant = 10;

if(!SetCommTimeouts(hcom, &tout))
{
   printf("\nError - SetCommTimeout\n");
   printf("%d\n", GetLastError());
   return (-1);
}

printf("\nSerial interface ok\n\n");
return (0);
```

This configuration causes a "zero wait state" in case there is no byte in the buffer when attempting to read. This gives you all the benefit of a quick program being able to act as a real time app without the demand of an extra thread. You will have to apply a kind of "polling" from within your program to make sure than all bytes are read. I installed an idle measuring mechanism in the reading part to count the number of "read fails". This number indicates a kind of "poll headroom" for your app.

The W32 Samples

To give beginners a starting point for their work, I want to provide some simple examples on using the W32-API for serial communications. The samples come as source files (text included in this document) but are also available as complete projects for the DEV-C++ environment. See the website for details.

The Port Sample

Description

The port sample was derived from a little machine control program and shows simple output and input operation. At the beginning, some characters are sent out for configuring, and incoming characters are monitored constantly. They are not processed in any way – so as long as you type sensible ASCII characters from within an attached Hyperterminal (second PC or looped) you should be able to read the characters.

Function

The program consists of a global loop which is entered immediately after the setup code and repeatingly induces a read and write routine. The program is theoretically capable to send and receiva data the same time. However, due to operation commands, sending is only performed at the beginning.

The Multiport Sample

Description

The multiport sample is an enhanced version of the single port sample, monitoring up to 16 channels. In this particular example, the "multiplication" is done my pultiple polling rather than threaded programming. We will come to this later in the "multithread sample". Polling the ports in a given order has some advantages.

Function

The Multithread Sample

… will demonstrate a multi threaded application, using the same code for multiple instances to access ports. This ist not yet included – sorry.

Serial Communications with wxWin

This chapter specially deals with writing applications using the wxWindows-Library and the CTB-Library for serial communication.

The CTB serial library

Well, wxWindows currently has no explicit support for serial communications, however. But there is an independant thirdparty library which easily can work together with wxWin. It has been created by Joachim Buermann and can be found on http://www.iftools.de/ .

Creating the library

First, you must create the library for your system before you can use it together with e.g. wxWindows or run the samples. There is a makefile included to build the library. You should import it to your Visual Studio in order to build the library from within the IDE. This also ommits problems with incorrect seetings of the visual studio variables (VSVARS32.BAT).

Importing a makefile in Visual Studio

This installation hint describes how to obtain a workspace from the given makefile.vc :

Open the "makefile.vc" and replace the "xcopy" command in the original makefile by "copy" to prevent an error which occurs, when you do not have xcopy on your system. Save as "makefile.mak". This saves one step of modifying the build settings below. You might also delete the last commands, which delete the uncopied library from the temporary path after the built. If you don't do this, you won't have a lib if the copy action fails.

From menu [file] select "Open workspace", browse the "make files" *.mak and select the modified makefile. A Dialog appears, telling you something like "makefile is not a studio file ...", "new project will be created to wrap makefile" - or so. In the dialog box, select "Win32" and save as "ctb." The required extension "dsp" will be added automatically by the wizard.

In the new Project set :

[BUILT],[set active configuration] : -> "CTB.WIn32 - Release"

[Project],[Settings] :

(General)->"Release"	(General)->"Debug"
"NMAKE /f makefile.mak"	"NMAKE /f makefile.mak"
"/a"	"/a"
"ctb.dll"	"ctbd.dll"
"ctb.bsc"	"ctbd.bsc"
(Debug) ->"Release"	(Debug) ->"Debug"
-none-	"ctbd.dll"

After having saved the project, you are ready to build your library !

Building the CTB - Library

Save the project and select [build][build ctb.dll] to build your library. During the process, you should obtain something similar to this :

```
--------------------Configuration: makectb60 - Win32 Release--------------------
Microsoft (R) Program Maintenance Utility   Version 6.00.8168.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.
 cl /Yd /Zi /c /D__WIN32__ /I ..\..\..\contrib\include  /Foexpect.obj /c expect.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.
expect.cpp
 cl /Yd /Zi /c /D__WIN32__ /I ..\..\..\contrib\include  /Foiobase.obj /c iobase.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.
iobase.cpp
 cl /Yd /Zi /c /D__WIN32__ /I ..\..\..\contrib\include  /Fomatch.obj /c match.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.
match.cpp
 cl /Yd /Zi /c /D__WIN32__ /I ..\..\..\contrib\include  /Foscan.obj /c scan.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.
scan.cpp
 cl /Yd /Zi /c /D__WIN32__ /I ..\..\..\contrib\include  /Fowin32\getopt.obj /c win32\getopt.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.
getopt.cpp
 cl /Yd /Zi /c /D__WIN32__ /I ..\..\..\contrib\include  /Fowin32\serport.obj /c win32\serport.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.
serport.cpp
 cl /Yd /Zi /c /D__WIN32__ /I ..\..\..\contrib\include  /Fowin32\timer.obj /c win32\timer.cpp
Microsoft (R) 32-bit C/C++ Optimizing Compiler Version 12.00.8804 for 80x86
Copyright (C) Microsoft Corp 1984-1998. All rights reserved.
timer.cpp
 lib winmm.lib expect.obj  iobase.obj  match.obj  scan.obj  win32\getopt.obj  win32\serport.obj  win32\timer.obj
/OUT:ctb.lib
Microsoft (R) Library Manager Version 6.00.8447
Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

---------------------------------------
```

If you get an error concerning "copy", please refer to the makefile.mak and see the particular paths and files. Eventually, copy the header files manually to the lib path.

In "C:\wx2\contrib\src\ctb" or "C:\wx2\lib" there should be your "ctb.lib".

Anyway :

Make sure that finally, the lib is present in the wxWindows lib directory and all header files are in the appropriate directory. You can check an existing library directly after built by compiling the sample.

See -> WTERM - Sample.

Modifying the Sources

As far as not already done in the latest release, you will have to make some changes to the CTB sources or header files :

More Port Names

Please read the serial programming techniques and the multiport chapter before, in order to be able to understand the meaning of these modifications. You possibly won't need this, if you do not make use of a multi port server or other special hardware and focus on the "wxCOM1" and "wxCOM2" only.

Since we want to access a number of possible new ports, named the common way, it is usefull to enlarge the table in the file : "serport.h"

Windows :	**Linux :**
/include/wx/ctb/win32/serport.h	/include/wx/ctb/linx/serport.h
#define wxCOM1 "com1" #define wxCOM2 "com2" #define wxCOM3 "com3" #define wxCOM4 "com4" **#define wxCOM5 "//./com5"** **#define wxCOM6 "//./com6"** **…** **#define wxCOM15 "//./com15"** **#define wxCOM16 "//./com16"**	#define wxCOM1 "/dev/cua0" #define wxCOM2 "/dev/cua1" #define wxCOM3 "/dev/cua2" #define wxCOM4 "/dev/cua3" **#define wxCOM5 "/dev/cua4"** **#define wxCOM6 "/dev/cua5"** **…** **#define wxCOM15 "/dev/cua14"**

It is not in recommended to include also the ports if special names are used.

Larger Buffer Size

The buffer size given in the "serport.cpp" (Win32) can be set to higher values in order to be ready for higher transfer rates

```
if(!SetupComm(fd,8192,8192))
```

More Baudrate Values

You should enlarge the arrays of baudrates in the subsequent by the desired values. In my case, the additional lower values were necessary to be compatible with some scanners and bar code readers, where the higher values are used to obtain the maximum out of the driver / physical port. Please note, that some hardware physically might not support certain transfer rates. Please note, that the file "serportx.h" is a common file for all systems.

Common :	**Windows :**	**Linux :**
src/ctb/serportx.h	src/ctb/win32/serport.cpp	src/ctb /linux/serport.cpp
/*! 150 baud */ **wxBAUD_150=150,** **/*! 300 baud */** **wxBAUD_300=300,** **/*! 600 baud */** **wxBAUD_600=600,** **/*! 1200 baud */** wxBAUD_1200=1200, /*! 2400 baud */ … /*! 57600 baud */ wxBAUD_57600=57600, /*! 115200 baud */ wxBAUD_115200=115200 **/*! 230400 baud */** **wxBAUD_230400=230400** **/*! 460800 baud */** **wxBAUD_460800=460800** **/*! 921600 baud */** **wxBAUD_921600=921600**	**case wxBAUD_150:** **baud = CBR_150;** **break;** **case wxBAUD_300:** **baud = CBR_300;** **break;** **case wxBAUD_600:** **baud = CBR_600;** **break;** **…** case wxBAUD_57600: baud = CBR_57600; break; case wxBAUD_115200: baud = CBR_115200; break; **case wxBAUD_230400:** **baud = CBR_230400;** **break;** **case wxBAUD_460800:** **baud = CBR_460800;** **break;** **case wxBAUD_921600:** **baud = CBR_921600;** **break;**	**case wxBAUD_150: return B150;** **case wxBAUD_300: return B300;** **case wxBAUD_600: return B600;** case wxBAUD_1200: return B1200; case wxBAUD_2400: return B2400; case wxBAUD_4800: return B4800; case wxBAUD_9600: return B9600; case wxBAUD_38400: return B38400; case wxBAUD_57600: return B57600; case wxBAUD_115200: return B115200; **case wxBAUD_230400: return B230400;** **case wxBAUD_460800: return B460800;** **case wxBAUD_921600: return B921600;**

Once you have modified sources for the library, you will have to rebuild it of course.

To enable the compiler to recognize the new transfer rates, given in the second column, you should also modify the refering header in the compilers include files !

See "\\Visual Studio\VC98\Include\WINBASE.H" :

```
#define CBR_110          110
#define CBR_150          150
#define CBR_300          300
#define CBR_600          600

…

#define CBR_38400        38400
#define CBR_56000        56000
#define CBR_57600        57600
#define CBR_115200       115200
#define CBR_128000       128000
#define CBR_256000       256000
#define CBR_230400       230400
#define CBR_460800       460800
#define CBR_921600       921600
```

The CTB Samples

On his website, Joachim Buermann offers sample programs, which make use of both the CTB library and wxWindows. (The WTERM is allready included in the CTB-sources)

The WTERM Sample

One of the sample is the WTERM example. It gives you two windows, to send data from one to another, or use even two instances of the program to send data vice versa.

Building the WTERM Sample

Import the given makefile into VS as shown before in this document. Without changes to the project settings, you can select “built makefile.exe” from the menu [BUILT]. You will get a program called “WTERM.EXE” in your main directory. Since it is not placed in the directory “Release” you can’t start it from the IDE – select it from exporer instead. To work with the WTERM, you should fully integrate it VS by making the appropriate project settings.

VisualStudio’s output could look like this :

```
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.
 cl @C:\DOKUME~1\JSCHUH~1\LOKALE~1\Temp\nma01400.
main.cpp
 cl @C:\DOKUME~1\JSCHUH~1\LOKALE~1\Temp\nmb01400.
console.cpp
 cl @C:\DOKUME~1\JSCHUH~1\LOKALE~1\Temp\nmc01400.
settings.cpp
 copy %WXWIN%\lib\ctb.lib .
        1 Datei(en) kopiert.
 rc -r /iC:\wx2\include /iC:\wx2\contrib\include -fowxterm.res wxterm.rc
 link @C:\DOKUME~1\JSCHUH~1\LOKALE~1\Temp\nmd01400.
LINK : warning LNK4098: defaultlib "LIBC" conflicts with use of other libs; use /NODEFAULTLIB:library

makefile1.exe - 0 error(s), 1 warning(s)
```

The WTERM should work on WinXP directly. When running on my Win2000-PC, the library “MSVCRTD.DLL” was reported missing. I copied it from XP into W2000 and it worked.

Modifying the WTERM Sample

Once you have more baudrates present in your headers, you can enhance the WTERM-Sample to use them. Open “settings.cpp” and add the following lines

```
static wxString baudrates[] = {
   "150",
   "300",
   "600",
   "1200",
   "2400",
   "4800",
   "9600",
   "19200",
   "38400",
   "57600",
   "115200",
   "230400",
   "460800" // not used in my app
};
```

Remarks :

At this point of time 460800 BAUD is possible but practically left out, since it is not supported by the multiport device.

You must also adjust the value for the number of menu entries in the subsequent code. In our example, we must set “13”.

```
   hs1->Add(new wxChoice(this,
                                    -1,
                                    wxDefaultPosition,
                                    wxDefaultSize,
                                    13,
                                    baudrates,
```

The same can be done with more ports :

```
static wxString ports[] = {
   "//./COM1",
   "//./COM2",
   "//./COM3",
   "//./COM4",
   "//./DM01P01",
   "//./DM01P02",
   "//./DM01P03",
   "//./DM01P04",
   "//./RP01P01",
   "//./RP01P02",
   "//./RP01P03",
   "//./RP01P04"
};
```

And also adjust the menu definitions some line below ...

```
   hs2->Add(new wxChoice(this,
                                    -1,
                                    wxDefaultPosition,
                                    wxDefaultSize,
                                    12,
                                    ports,
```

Remark 1 : However, the above modifications do work, disregarding the programming issue “slash problem” , shown in the W32 part.

Remark 2: If you have modified the CTB-sources and rebuilt the lib, you might also use the more common declarations given in the CTB-headers :

wxCOM1, wxCOM2 … wxCOM16

(I did not try this so far)

Working with the WTERM Sample

Here are some examples of applications and configurations. I used Hyperterminal, WCOM, and WTERM itself to establish a communication. WCOM ist an application for serial port testing and analyzing provided by the multi port manufacturer.

Loopback with one PC :

Direct conncection of two PCs with cross over cable :

Hardware-Config :	PC1-COM1-cable-COM1-PC1
Settings used :	9600 – 460k, 8 bit, No Parity, 1 Stop Bit, no flow control

WTERM <–> WTERM

WTERM ist used by pasting strings into the lower window expecting it to appear in the upper window of the receiveing WTERM application.

a) Pasting a 1-80 character string and sending by CR makes the string appear directly
b) Pasting an e.g. 100 char string, is a bit slower , however.

It is also noticable, that significant delays occur, when a new line has to be drawn.

All types of transfer rates (9,600 – 460k) worked.

The major issue hereby is, that you can interlink the PCs with theoretically 460k speed ! – But according to my observation this works only with small string sizes.

3. WCOM - WTERM

A quick continuous stream send with WCOMs automatic testdata causes bad characters. Restarting WTERM helps. Also stopping send actions and restarting after a pause.

This happens already at 57k transfer rate. The first lines of testdata are received correctly – but then a kind of buffer overrun seems to occur. Maybe also a matter of terminal update and data processing. Any time, the send process is started, at least (around) 6000 Bytes are read.

If problems occur change the Buffer settings in der Sources !

Transfer rates work up to 115k – higher values cannot be set due to “set commstate error !”

The SERTEST Samples

Another sample is the sertest sample.

This article is not yet finished.

The wxWin Samples

The Port Sample

Description

Function

This article is not yet finished.

The Multi Port Sample

Description

Function

This article is not yet finished.