8 Developers Installation

The purpose of an open source project is that other developers modify and enhance the existing framework. This section should provide a help to install and setup the environment to rapidly start with developing. The most annoying thing in developers work is spending a lot of time to install the programming environment. We try to make this step as easy as possible.

8.1 Resources

To develop on the current Java USB API project some resources are required:

JavaUSBComplete

JavaUSBComplete.ZIP
The Java USB API sources including the native libraries and the jUSB driver. This source can be downloaded on http://www.steelbrothers.ch/jusb/. Be sure you download the Java USB complete resources for developers which is called JavaUSBComplete.ZIP

Visual C++

Microsoft Visual C++ or the new version Microsoft Visual .NET
Microsoft Visual C++ or the new version Microsoft Visual .NET programming environment has to be used. We developed a big part of the Java USB API for Windows on Microsoft Visual C++ Version 6.0.

Platform SDK 2003

Microsoft Software SDK 2003 or later
The Microsoft software developer kit is needed to support the core libraries. The latest SDK can be downloaded from the MSDN developers site (http://www.microsoft.com/msdownload/platformsdk/sdkupdate/). Only the Core SDK is needed (168 MB). The required size for the complete installation is 480MB.

DDK XP

Microsoft Driver Development Kit (DDK) XP or 2003
The DDK is used to build the JUSB driver. Unfortunately, the DDK is not made available for download by Microsoft. An order is demanded, but they will send the DDK for free apart of the expenses for delivery. The DDK page can be found at the following url: http://www.microsoft.com/whdc/ddk/.

External Debugger

DbgView
We used the debug view program DbgView which can be freely downloaded from http://www.sysinternals.com. This program intercepts calls made to *DbgPrint* and *KdPrint* by device drivers and *OutputDebugString* made by Win32 programs. It allows for viewing and recording of debug session output on your local machine or across the Internet without an active debugger [25]. This tool has been a big benefit for developing either user or kernel mode programming.

Netbeans

Netbeans IDE
Is a full-featured integrated environment for Java Developers [21]. We used Netbeans to develp the Java side of the Java USB API. Netbeans is freely available from http://www.netbeans.org. Of course any other environment with a Java compiler can be used to extend or run the Java USB API.

JDK 1.4.1

Java Runtime Environment J2SE 1.4.1
The premier solution for rapidly developing and deploying mission-critical, enterprise applications, J2SE provides the essential compiler, tools, runtimes, and APIs for writing, deploying, and running applets and applications in the Java programming language [9]. We compiled the Java USB API with the JDK version 1.4.1 [8].

8.2 Setting the Environment Variables

Different kinds of environment variables have to be adjusted. Some of them are

specific for the Netbeans IDE and others for settings in the Visual C++ project. If those environment variables are not set correctly, parts of the software environment will not work correctly. Table 32 shows where the environment variables can be set.

Environment Variables

Windows 2000

1. Start→Settings→Control Panel→System
2. choose category: Advanced
3. choose: Environment Variables…
4. Edit, delete or make a new system variable

Windows XP

1. Start→Control Panel→System
2. choose category: Advanced
3. choose: Environment Variables
4. Edit, delete or create a new system variable

Table 32: Setting the environment variables

The following environment variables have to be set. The value given in this context can be seen as an example. This belongs to the settings we specified in our computer and they will vary on other system. The example should give a hint of what the path may look like.

CLASSPATH

CLASSPATH

The class path tells SDK tools and applications where to find third-party and user-defined classes - that is, classes that are not Java extensions or part of the Java platform. The class path needs to find any classes you have compiled with the javac compiler - its default is the current directory to conveniently enable those classes to be found. We may need to extend the CLASSPATH variable, because other settings are defined too. Extension are made by a semicolon '**;**'.

Variable	Value
CLASSPATH	F:\Studium\JavaUSB\JavaSources

Table 33: CLASSPATH setting

Path

Path

While trying to compile a Java source with javac or creating a JNI header file with javah and receiving the following error message: *'javac' is not recognized as an internal or external command* or *'javah' not found* then the Path variable need to be set to the path where the binaries of those commands are. This variable needs in all probability to be extended.

Variable	Value
Path	C:\Programme\s1studio_jdk\j2sdk1.4.1_02\bin;

Table 34: Path setting

JAVAHOME

JAVAHOME

This variable points to the root directory of the Java runtime environment. This setting enables the Visual C++ programming environment to find the Java Native Interface header files, such as jni.h.

Variable	Value
JAVAHOME	C:\Programme\s1studio_jdk\j2sdk1.4.1_02

Table 35: JAVAHOME setting

JUSBPATH

JUSBPATH

This variable points to the JavaUSB directory. This setting is used when a developer does not have the DDK [6] installed but still wants to try compiling and

JUSBPATH

modifying the JUSB DLL. This provides the Visual C++ project settings where to find the additional header file which would have been on the DDK from Microsoft. These DDK header file we use can be found in the *JusbDll\external-header-file\ddk* folder.

Variable	Value
JUSBPATH	F:\Studium\JavaUSB

Table 36: JUSBPATH setting

DDKPATH

DDKPATH
Points to the root directory of the current installed DDK.

Variable	Value
DDKPATH	C:\WINDDK\2600.1106

Table 37: DDKPATH setting

8.3 Unzip the JavaUSBComplete.Zip File

To work with the JUSB DLL the JavaUSBComplete.Zip file needs to be extracted. For further examples we assume that the JavaUSBComplete.Zip file is unzipped in a folder, named *JavaUSB*.

Make sure not to copy the *JavaUSB* folder within a folder path containing spaces in the name for example *"C:\Documents and Setting\..."*. This will lead to error in the build environment for the driver (see 8.6.1.1).

After successfully unzipping the JavaUSBComplete.Zip file, four folders should be seen in the JavaUSB folder:

- *Installation Files*: Contains all files for end users that want to use the Java USB API in application.
- *JavaSources* : Complete Java source code of the Java USB API. Chapter 8.4 gives an overview of the folder contents.
- *JusbDll*: All C/C++ source files which are used to create the JUSB DLL and the JNI implementation of the Java USB API. Chapter 8.5 will explain the contents of this folder in detail.
- *JusbDriver*: Driver relevant resources to build the driver. Chapter 8.6 presents al the files belonging to the JUSB driver and how the driver can be built.

8.4 Java USB API for Windows

All the needed files to implement or extend the Java USB API can be found in the usb.windows package which is in the *\JavaSources\usb\windows* folder. As development environment any text editor can be used or Netbeans IDE as we did. Make sure that you start compiling the classes from the root directory of the package. For example if we want to compile the Windows class, we need to be in the *JavaSources* directory. Run the command line program and enter the following command: javac usb.windows.Windows.java.

8.4.1 Creating the Java Native Headers

javah

To implement the native methods which are specified in the Java classes, we need to create the appropriate C-header files. This is done in using the command **javah**. Suppose we add a new native method to the *JUSB* class and want to create the C-header file then use the following command:

```
javah –jni usb.windows.USB
```

The corresponding header file usb_windows_USB.h is put into the Java root directory, which should be *JavaSources.* Remember while creating the JNI

header file that we always have to write the whole package name of the class and need to call the command from the Java root directory. If we disregard this restriction and call javah just in the current Windows directory then we would get a JNI header file named as USB.h. The information about the package is lost and this leads to error while loading the JUSB DLL [3].

8.4.2 Directory and File Description

We describe only the files belonging to the Windows package. For the description of the whole Java USB project refer to [18].

The usb.windows package files are in the following directory path: \JavaSources\usb\windows. Table 38 lists those files.

Filename	Description
DeviceImpl.java	Implements the class to which all USB devices belongs to either running as a JUSB device or not.
JUSB.java	Contains methods that can be used for devices that are adapted for the JUSB driver. This class does implement the DeviceSPI class.
NonJUSB.java	Does implement the DeviceSPI method in throwing only IOExceptions otherwise it does nothing else.
USB.java	This class implements the Bus class and provide access to a USB bus.
USBException.java	Contains the USBExceptions that are thrown when having a USB specific error.
Windows.java	This class implements a singleton USB host.

Table 38: Files in the usb.windows package

8.5 jUSB DLL

Installation of JUSB DLL

This section provides information for developer interested in extending the jUSB DLL for the Java USB API. There is a little operating system version conflict between Windows 2000 and Windows XP that we encounter while creating the DLL on Windows 2000. All the development has been made on Windows XP Professional and a full installation of the DDK [6] and SDK [24]. To give developers on Windows 2000 the opportunity to extend this Java USB API a specific section introduce the different project settings in the Visual C++ environment. We recommend new developers to use Windows XP or higher environments for developing the Java USB API for Windows.

At first, the Microsoft SDK 2003 has to be installed on the computer.

Start the Visual C++ environment and open the workspace (File→Open Workspace…) jusb.dsw which can be found in the *JusbDll\jusb*\ folder. Before start editing and working on the files, we need to set the project setting depending on the operating system we are currently running. Next subsection is going to explain the project settings in detail. To do the setting in the Visual C++ environment choose Project→Settings and then choose the appropriate rubric.

8.5.1 Visual C++ 6.0 Project Setting

Some project settings need to be done to successfully build the JUSB dynamic link library. The setting depends on the Windows Version (2000 or XP) and if we have installed the driver development kits or not.

In all cases we need the proper installation of the Microsoft SDK 2003. To make sure the SDK is added to the Visual C++ project check (Tools→Options.. →Directories) check if the following entry "C:\Programme\Microsoft SDK\include" depending on where we installed the SDK is on the **top** by the include files.

8.5.1.1 Project Settings without the DDK

To make it possible to build the dynamic link library without the driver development kit from Microsoft, the used header file from the DDK are made available in the following folder: *JusbDll**external-header-file**ddk*.

Attention of the Environment Variables

The next paragraph describes the project settings for Windows 2000 and XP. Because we use environment variables, we have to be aware of one important fact for the settings. Usually the environment variable are used in the project settings as follows:

```
$(JUSBPATH)\JusbDll\external-header-file\java\include\
```

but if the environment variable contains spaces in the string, for example JUSBPATH defined as "C:\Documents and Settings\JavaUSB" then we need to quote the entry in the project settings!

```
"$(JUSBPATH)\JusbDll\external-header-file\java\include\"
```

If we do not care about this fact, the compiler will not build the DLL and ends with an error like : *error LNK 1104: cannot open file "Documents.obj"*.
Thereby belongs the Documents.obj file to the the name of C:\Documents where in fact does not exist. To make sure that such an error does not occur, choose an environment variable value with no spaces within the path or quote all the project settings where an environment variable is used.
In the following example we assume to have environment variable values without any spaces within the string.

Project Settings in Windows 2000 without the DDK

8.5.1.2 Windows 2000

Add the SDK to Options as described in 8.5.1 and then set the following project settings as in Table 39.

Rubric **C/C++**: Category: Preprocessors **Additional include directories:** *$(JUSBPATH)\JusbDll\external-header-file\java\include\,* *$(JUSBPATH)\JusbDll\external-header-file\java\include\win32\,* *$(JUSBPATH)\JusbDll\external-header-file\ddk\inc\,* *$(JUSBPATH)\JusbDll\jni\,* ***$(JUSBPATH)\JusbDll\external-header-file\ddk\inc\w2k*** Rubric **Link** Category: General **Object/Library modules:** (add the following entries to the existing) ***$(JUSBPATH)\JusbDll\external-lib-file\w2k\setupapi.lib*** *$(JUSBPATH)\JusbDll\external-lib-file\hid.lib*

Table 39: Project settings in Windows 2000 without DDK

Comment Out Functions

If we now build the JUSB DLL we get some error of SPDRP_XXX undeclared identifier. This happens because we tried to use the new SetupDiXxx API function to retrieve registry information. Windows 2000 does not support completely those functions. Therefore we need to comment out 3 functions in the helperFunction.cpp file. These are:

- getRegistryPropertyString
- doDriverNameToDevicesDesc

- getRegistryInfo

Further we need to comment out the three function prototype in the jusb.h file.
We do not use these function either in Windows XP, but they are already coded to be used for future work on the Java USB API.
After having done those changes the JUSB DLL should be built without any errors.

Project Settings in Windows XP without DDK

Windows XP
The jusb.dsw project contains all the current settings and therefore no additional settings should be necessary. The following setting should be predefined:

Rubric **C/C++**: Category: Preprocessors **Additional include directories:** *$(JUSBPATH)\JusbDll\external-header-file\java\include\,* *$(JUSBPATH)\JusbDll\external-header-file\java\include\win32\,* *$(JUSBPATH)\JusbDll\external-header-file\ddk\inc\,* *$(JUSBPATH)\JusbDll\jni* Rubric **Link** Category: General **Object/Library modules:** (add the following entries to the existing) *setupapi.lib* *$(JUSBPATH)\JusbDll\external-lib-file\hid.lib*

Table 40: Project settings in Windows XP without DDK

8.5.1.3 Project Settings with an Installed DDK

If the DDK is installed, it does not make sense to use the DDK header provided in the *JavaUSB* folder and is appropriated to use the original DDK header files. The paragraph “Attention to environment variables” in 8.5.1.1 still has its validity.

Project Settings in Windows 2000 with DDK

Windows 2000
Next to the project settings there must be done additional changes in different source files. First, set the project settings according to Table 41.

Rubric **C/C++**: Category: Preprocessors **Additional include directories:** *$(JAVAHOME)\include\,* *$(JAVAHOME)\include\win32\,* *$(DDKPATH)\inc\w2k\,* *$(DDKPATH)\inc\ddk\w2k\,* *$(JUSBPATH)\JusbDll\jni* Rubric **Link** Category: General **Object/Library modules:** (add the following entries to the existing) *$(DDKPATH)\lib\w2k\i386\setupapi.lib* *$(DDKPATH)\lib\w2k\i386\hid.lib*

Table 41: Project settings in Windows 2000 with the DDK installed

Function need to be out documented in Windows 2000

If we now build the JUSB DLL we get some error of the form SPDRP_XXX undeclared identifier. This happens because we tried to use the new SetupDiXxx API function to retrieve registry information. Windows 2000 does not completely support those functions. Therefore, we need to comment out 3 functions in the helperFunction.cpp file. These are:

- getRegistryPropertyString
- doDriverNameToDevicesDesc

- getRegistryInfo

Further, we have to comment out the three function prototypes in the jusb.h file. We do not use these function either in Windows XP, but they are already coded to be used for future work on the Java USB API.

New Constants definition in jusb.h running on Windows 2000

We also need to activate the comment out definition in jusb.h for bmRequest.Dir, bmRequest.Type and bmRequest.Recipient which will be found almost on the top. The reason for that definition is because we already used those constants which are defined in usb100.h in the DDK. The DDK file usb100.h in Windows 2000 does not define those constants. In the newer version of DDK XP they are defined in usb100.h. Table 42 shows the correct settings for Windows 2000.

```
// Only used when running on Windows 2000!
//bmRequest.Dir
#define BMREQUEST_HOST_TO_DEVICE       0
#define BMREQUEST_DEVICE_TO_HOST       1

//bmRequest.Type
#define BMREQUEST_STANDARD            0
#define BMREQUEST_CLASS               1
#define BMREQUEST_VENDOR              2

//bmRequest.Recipient
#define BMREQUEST_TO_DEVICE           0
#define BMREQUEST_TO_INTERFACE        1
#define BMREQUEST_TO_ENDPOINT         2
#define BMREQUEST_TO_OTHER            3
```

Table 42: Definition of bmRequest constants only under Windows 2000

Out document some line in getAttachedDeviceType function

Furthermore, we need to comment out some *else* branches in the *getAttached-DeviceType* function in jusb.cpp. The reason for that is the modification from the usbioctl.h header file in the DDK. The USB_CONNECTION_STATUS enumeration type has been extended with two members (DeviceHubNestedTooDeeply and DeviceInLegacyHub). Those members are not known in the Windows 2000 environment and we therefore have to comment out those lines as shown in Table 43.

```
int getAttachedDeviceType(HANDLE hubHandle, int portIndex){
…
  if(…){
  …
  }else if(connectionInfo.ConnectionStatus[0] == DeviceNotEnoughBandwidth){
   return -5;
  }else /* if(connectionInfo.ConnectionStatus[0] == DeviceHubNestedTooDeeply){
   return -6;
   }else if(connectionInfo.ConnectionStatus[0] == DeviceInLegacyHub){
    return -7;
   else */ if(connectionInfo.ConnectionStatus[0] == DeviceFailedEnumeration){
   return -8;
   }else return 0;
…
}
```

Table 43: Modified getAttachedDeviceType function (Windows 2000)

After having done those changes, the JUSB DLL should be built without any errors.

Windows XP

Table 44 shows all the additional settings.

Project Settings in Windows XP with the DDK

Rubric **C/C++**: Category: Preprocessors

Project Settings in Windows XP without the DDK	**Additional include directories:** *$(JAVAHOME)\include\,* *$(JAVAHOME)\include\win32\,* *$(DDKPATH)\inc\wxp\,* *$(DDKPATH)\inc\ddk\wxp\,* *$(JUSBPATH)\JusbDll\jni* Rubric **Link** Category: General **Object/Library modules:** (add the following entries to the existing) *setupapi.lib* *$(DDKPATH)\lib\wxp\i386\hid.lib*

Table 44: Project settings in Windows XP with installed DDK

8.5.2 Directory and File Description

All the files required to build the jUSB DLL are available in the *JusbDll* folder. The folders within the *JusbDll* folder are listed in Table 45.

Foldername	Description
jusb	Contains all C++ source files, C-header files and all the files to create the project workspace for the Visual C++ environment.
jni	Contains all JNI header files that are created with javah. The header files are named automatically when executing javah. These header files should not be modified. If new native methods have been added to a Java class, run javah with the modified Java class to acquire the corresponding JNI header file, and copy that header file if necessary in that folder.
external-lib-file	Contains libraries which are used when no DDK is installed (see 8.5.1.1).
external-header-file	Contains header files that are used when no DDK is installed (see 8.5.1.1).

Table 45: Folders in JusbDll Folder

The files in the *jni*, *external-lib-file* and *external-header-file* folder are not explained in detail. The important files for developers are in the *jusb* folder which will be described in the following paragraphs.

Figure 18 provides an overview about the files which are related to each other. The general framework decision was to provide for every automatically created JNI header file its own file where the implementation is done. Furthermore, the decision was made to keep the function code within these implementation files as short as possible to keep it readable. If a function implementation becomes more complex, an external function was made to process the desired request. These external functions are put in the jusb.cpp, helperFunction.cpp and devnode.cpp files. Figure 18 represents this partitioning in the block B. All three blocks A, B and C represent one function body.

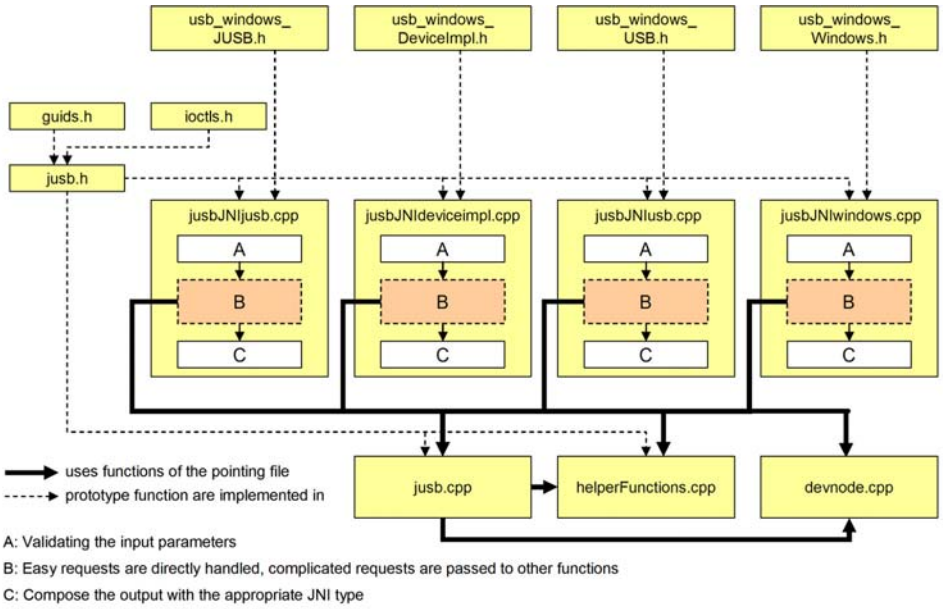

Figure 18: File composition of jUSB DLL project

Filename	Description
guids.h	Contains the GUID for GUID_DEFINTERFACE_JUSB_DEVICES which the jUSB driver registers when a device uses the JUSB driver. With the aid of this GUID we are able to locate JUSB devices. More about GUID can be found in Appendix X .
ioctls.h	Contains all the IOCTL codes which are available in the jUSB driver.
jusb.h	Definition of structures, variables and constants. Further, it contains also all function prototypes that are used in more than one file.
devnode.cpp	Contains one function that retrieves the DeviceDesc registry entry from a given driver name. This file may be obsolete for future work and should be replaced with the new registry function, such as getRegistryPropertyString, getRegistryInfo and doDriverNameToDeviceDesc which are already implemented in the helperFunctions.cpp.
helperFunctions.cpp	Contains function to process complicated requests.
jusb.cpp	Is the entry point for the DLL. It contains also function to process complicated requests.
jusbJNIdeviceimpl.cpp	Implements the JNI function of the Java *DeviceImpl* class.
jusbJNIjusb.cpp	Implements the JNI function of the Java *JUSB* class.
jusbJNIusb.cpp	Implements the JNI function of the Java *USB* class.
jusbJNIwindows.cpp	Implements the JNI function of the Java *Windows* class.
jusb.dsw	Visual C++ Workspace
jusb.dsp	Visual C++ Project

jusb.html jusb.mak jusb.opt jusb.ncb	The next four files belonging to the Visual C++ environment.

Table 46: Descriptions of files in the jusb folder

8.6 JUSB Driver

This section describes all parts of the jUSB driver. Useful information is provided to build and develop the jUSB driver. It is absolute necessary that the SDK [24] and the DDK [6] from Microsoft is installed and the relevant environment variables are correctly set.

The build process depends not on the operating system we develop. The procedure is identical in Windows 2000 and Windows XP, because we assume having installed on both operating systems the DDK XP version.

The Microsoft® Windows® Driver Development Kit (DDK) release for Microsoft® Windows® XP incorporates a number of significant design changes. The most important of these include: a new setup program, new build environments, a redesign of the layout of the installed headers and libraries, and new build tools that make the new DDK a stand-alone product. A feature has also been added to the build environment to help developers identify the use of deprecated functions in their code at build time [22].

8.6.1 How to Build the Driver

As introduction we present a section (Table 47) of the paper *New in the DDK for Windows XP* [22].

New Build Environment
A number of important changes have been made to the Windows DDK build environment. For one, the Windows DDK now includes a complete set of tools for building drivers. Microsoft® Visual C++® is no longer required to be installed to use the DDK. Use of the included tools for all Windows 2000 and Windows XP drivers is expected within the shipping build environment. This version of the Windows DDK does not support building Windows XP or Windows 2000 drivers using a version of Microsoft Visual C++ earlier than the one supplied with the DDK. Attempts to use an incorrect version of Visual C++ will result in the following error message from the compiler:

error C1189: #error : Compiler version not supported by Windows DDK

This requirement is due to the reliance on many new features within this tool set for proper functioning of the include build environment. The compiler, linker, and supporting files, such as C Run-Times (CRTs), should be considered an atomic unit within the build environment. It is likely that mixing the supporting tool files of Visual C++ versions 5 or 6 with those in this DDK release, which are based on the new Visual C++ version 7 code base, will result in errors in the binaries. Using the provided build environment and build tools is thus strongly recommended to ensure generation of valid binaries [22].

Table 47: Build environment from the DDK

The conclusion of the section in Table 47 is that we can not build the jUSB driver within Visual C++ Version 6.0 because the compiler version does not correspond to the DDK version. To compile the jUSB driver within Visual C++ environment, an update of Visual C++ is required (Visual .NET). We started developing the driver in Visual C++ environment and did not change the environment during the project. Therefore we build the driver with the build environment delivered within the DDK but still edit the code within the Visual C++ environment.

Building Steps

Build the jUSB driver:

1. Start the Win XP Checked Build Environment (also in Windows 2000) (Start→Programs→Development Kits→Windows DDK 2600.1106→ BuildEnvironments→Win XP Checked Build Environment)
2. Change the directory path so that it points to the *sys* folder which is a subfolder of *JusbDriver*. For example:
   - change the drive: **F:** <enter>
   - change the folder path: **cd JavaUSB\JusbDriver\sys** <enter>
3. Enter command: **build –cZ** <enter>
   Some states of the building process are printed on the output screen (an output example is shown in Table 48).
   The most important statement is : 1 executable built
   If this statement is missing check chapter 8.6.1.1 for more information
4. The compiled jUSB driver (jusb.sys) can be found in the following sub-folder of the *sys* folder: ***\objchk_wxp_x86\i386***
5. Copy the jusb.sys file into the *system32\drivers* folder of the Windows main directory.
   If there is already a registered jUSB driver in the directory, only a re-placement of the jsub.sys file needs to be done. Otherwise if it is the first time using the jUSB driver refer to chapter 7.2 (user installation) for more information about how to register the jUSB driver.

Screen Shot of Building Process

```
F:\Studium\JavaUSB\JusbDriver\sys>build -cZ
BUILD: Adding /Y to COPYCMD so xcopy ops won't hang.
BUILD: Object root set to: ==> objchk_wxp_x86
BUILD: Compile and Link for i386
BUILD: Examining f:\studium\javausb\jusbdriver\sys directory for files to compile.
BUILD: Building generated files in f:\studium\javausb\jusbdriver\sys directory
BUILD: Compiling f:\studium\javausb\jusbdriver\sys directory
Compiling - jusb.rc for i386
Compiling - driverentry.c for i386
Compiling - plugplay.c for i386
Compiling - power.c for i386
Compiling - control.c for i386
Compiling - wmi.c for i386
Compiling - readwrite.c for i386
Compiling - generating code... for i386
BUILD: Linking f:\studium\javausb\jusbdriver\sys directory
Linking Executable - objchk_wxp_x86\i386\jusb.sys for i386
BUILD: Done

   8 files compiled
   1 executable built

F:\Studium\JavaUSB\JusbDriver\sys>
```

Table 48: Output of jUSB driver build process

8.6.1.1 No Driver Executable Built

In case of missing the statement: "**1 executable built**" as shown in Table 48 check the **buildchk_wxp_x86.log** file. If the string *"'jvc' is not recognised as an internal or external command"* then your folder path to the *JusbDriver* folder contains somewhere spaces.

Spaces are not allowed in the driver path!

Solution: Copy the *JusbDriver* folder in a path with no spaces and do build the driver again as described in 8.6.1.

8.6.2 Directory and File Description

The files for the jUSB driver are all in the *JusbDriver\sys* folder. The following Table 49 lists all those file.

Filename	Description

Control.c	Implements the DispatchControl function which handle all I/O request packet (IRP) with function code IRP_MJ_DEVICE_CONTROL in the major field of the IRP.
Driver.h	Header file containing all definition of structs and global variables used in the driver.
DriverEntry.c	The entry point to the jUSB driver. Similar to a main file.
guids.h	The definition of the device interface, a global unique identifier named GUID_DEFINTERFACE-_JUSB_DEVICES
ioctls.h	The definition of IOCTL codes handled by the jUSB driver.
jusb.inf	The INF file used to register the jUSB driver to a device using an INF file.
jusb.reg	File to register the jUSB driver in the Windows registry.
jusb.bmf jusb.mof	Used in the makefile to build the jUSBdriver.
jusb.rc	Resources file containing information about the jUSB driver.
PlugPlay.c	Implements the DispatchPnP function which handles all I/O request packet (IRP) with function code IRP_MJ_PNP in the major field of the IRP.
Power.c	Implements the DispatchPower function which handles all I/O request packet (IRP) with function code IRP_MJ_POWER in the major field of the IRP.
ReadWrite.c	Implements the DispatchReadWrite function which handles all I/O request packet (IRP) with function code IRP_MJ_READ or IRP_MJ_WRITE in the major field of the IRP. This file is leftover from the bulkusb project and is not used so far in the jUSB driver. Some functions are used in other files and therefore this file has not been removed.
Wmi.c	Implements the WmiRegistration function.
All the other file in this folder belongs either to the project settings for Visual C++ or to the build process. There are not further described.	

Table 49: Files and its description in the JusbDriver folder