Interface of Direct IO DLL to Java® 1.2 or Higher

These are the programming guide lines required to interface the Module in Java as follows:

NOTE: System must contain JRE 1.2 or Higher for execution and J2SE for development.

1. **Directory Setup :** How to copy the files.?

   The files required for a project to use the DirectIO functionalities are provided in the "SetupPacks\JavaApp" folder. These files need to be copied to the project directory and some minor settings have to be done to include the DirectIO functionality in the project. The following are the files and corresponding target copy locations:

\SetupPacks\JavaApp\Directio.dll	\{your project folder}\Directio.dll
\SetupPacks\JavaApp\Porttalk.sys	\{your project folder}\Porttalk.sys
\SetupPacks\JavaApp\makeit.bat	\{your project folder}\makeit.bat
\SetupPacks\JavaApp\Meta-inf\Manifest.mf	\{your project folder}\Meta-inf\Manifest.mf
\SetupPacks\JavaApp\ PortAccess\HWAccess.class	\{your project folder}\ PortAccess\HWAccess.class

   NOTE: **wherever the project to function like an exe the generated .jar file must be used and JRE must be installed.**

2. **Make - Setup :** How to configure the build settings in **makeit.bat** ?

   This file *makeit.bat* is a batch file that can compile the program, make the .jar file and show the output. The following are its contents:

```
javac -classpath . {your File}.java > classerr.txt
jar cvfm pport.jar .\meta-inf\MANIFEST.MF {your File}.class
.\PortAccess\HWAccess.class .\directio.dll .\porttalk.sys > out.txt
java -jar {your File}.jar > err.txt
```

   The first line is for the java compilation and classer.txt will contain the errors. You may also include some of your java files if you require like:

```
javac – classpath . {your main file}.java app1.java app2.java > classer.txt
```

   The next line is for the jar file generation along with the class files.
   You can include your own class files also but you must in any case specify the class containing the "main()" or "init()" function in the MANIFEST.MF file explained later. The out.txt file states the processing of the jar archiving command and indicates if any error occurs. The other files can also be included E.g.:

```
jar cvfm pport.jar .\meta-inf\MANIFEST.MF {your File}.class app1.class app2.class
.\PortAccess\HWAccess.class .\directio.dll .\porttalk.sys > out.txt
```

   The last line is for execution of the java class in the Jar file and err.txt shows the command line output or errors if any.

   This file to be executed properly requires the path settings to be done in autoexec.bat for JDK like:

```
Set Path = %path%;{JDK dir path}\bin; {JDK dir path}\include; {JDK dir path}\lib
```

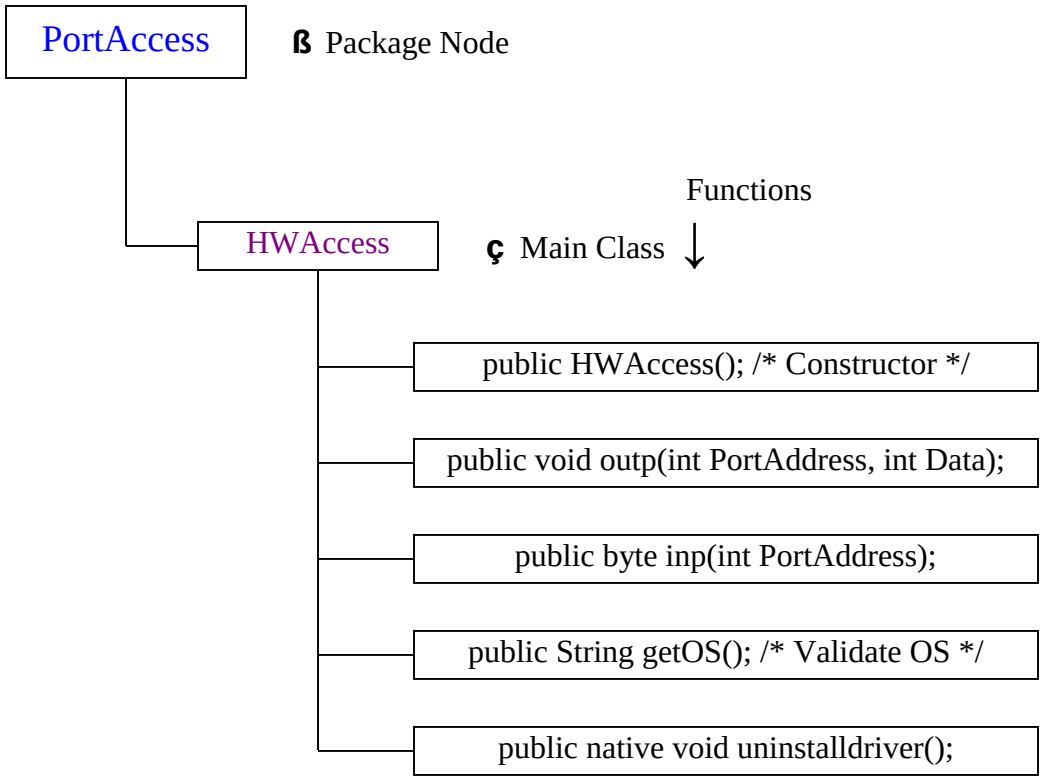
3. **Execution:** How to configure the execute settings in **MANIFEST.MF**?
   This *manifest.mf* file is used to specify the java class containing the "*main()*" or the "*init()*" function. These functions existing in any class means that class is the one which starts the "Application" or the "Applet". The manifest.mf file is responsible for specifying this particular class. The contents of the Manifest.mf file provided in the SetupPacks is as follows:

```
Manifest-Version: 1.0
Created-By: <Your Name>
Main-Class: <Your Class Containing main or init startup>
```

   Where the "Created-By" field should contain name of the author without ":" or "<>" characters. And the "Main-Class" field should contain the case sensitive filename of the ".class" file containing the "main()" or "init()" functions without the extension {like if app1.class is the file containing the "main()" or "init()" then Main-Class: app1}. This is clearly given in the Java Application "pport" provided in the examples.

Package Description and Classes:

This package has the following structure :



Member Functions and Descriptions:

1. `public HWAccess();`

   This is the constructor for the HWAccess class used to create the objects of this class. A constraint for this class is that it can be instantiated only once or it requires the Garbage Collector to be initiated.

   Use: `HWAccess io = new HWAccess();`

2. `public void outp(int PortAddress, int Data);`

   This function is the main 'output to port function' at a particular address . There are two parameters : the First one is the PortAddress and the Second one is the DATA to be sent. The parameter ranges are :

   ```
   Address : 0x0000 [or &H0000 or 0000H] to
             0x0FFF [or &H0FFF or 0FFFH] in Hexadecimal.
              {12 bit Parameter}

   DATA    : 0x000 [or &H000 or 000H] to
             0x0FF [or &H0FF or 0FFH] in Hexadecimal.
              { 8 bit Parameter}
   ```

   Use: `io.outp(0x378,0xAB);`

3. `public byte inp(int PortAddress);`

   This function is the main 'input from port function' at a give port PortAddress. There is only one parameter that is the address of the port. The DATA read is returned back by the function. The parameter range is:

   ```
   Address : 0x0000 [or &H0000 or 0000H] to
             0x0FFF [or &H0FFF or 0FFFH] in Hexadecimal.
              {12 bit Parameter}

   The returned value ranges are :
   DATA    : 0x000 [or &H000 or 000H] to
             0x0FF [or &H0FF or 0FFH] in Hexadecimal.
              { 8 bit Parameter}
   ```

   Use: `public byte in_val= io.inp(0x378);`

4. `public native void uninstalldriver();`

   This function is to uninstall the system Kernel Mode driver porttalk.sys from the system and deregister the service. This function is required to be executed for uninstall in case if the OS is an NT/2000/XP.

   Use: `io.uninstalldriver();`

5. `public String getOS();`

This Function returns the Version of any Microsoft Operating Systems Some string as follows:

```
{win3.11}       = "Windows 3.11"
{WinNT 3.5}     = "Windows NT 3.50"
{Win 98 or 95} = "Windows 9x"
{WinNT 4.0}     = "Windows NT 4.00"
{Windows2000}  = "Windows 2000"
{WindowsXP}     = "Windows XP"
{For any new OS like LongHorn}
                = "Windows LongHorn/New Shell"
{For any error}= "No Correct OS Found"
```

Use: `public String currentOS= io.getOS();`