IS502052: Enterprise Systems Development Concepts

Lab 5: Hello World Application with EJB

I. Introduction

In this lab, we will concentrate on implementing a very beginning application with EJB, which implementing both server and client sides by using a stateless session bean.

II. Steps to be Followed

In theory, to implement an application with EJB, you should follow these steps:

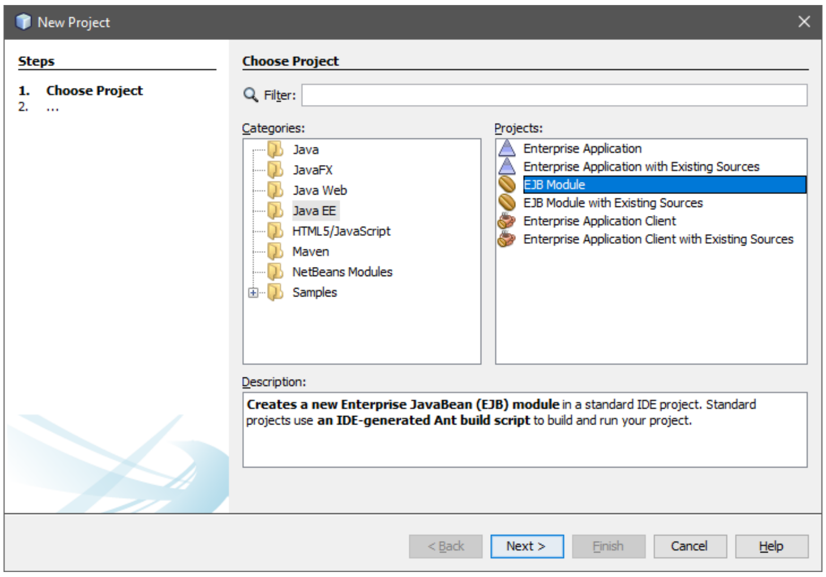
- Component development: describe ***Remote*** interface, describe ***Home*** interface, implement the ***Bean*** class.
- Write deployment descriptor(s)
- Package in a *jar* archive all EJB files
- Deployment into the **container**
- Implement the ***client*** application

III. Programming EJB Application in NetBeans

Technically, we will use NetBeans IDE to work with EJB. Let's begin implementing your first EJB application!

1. Create project

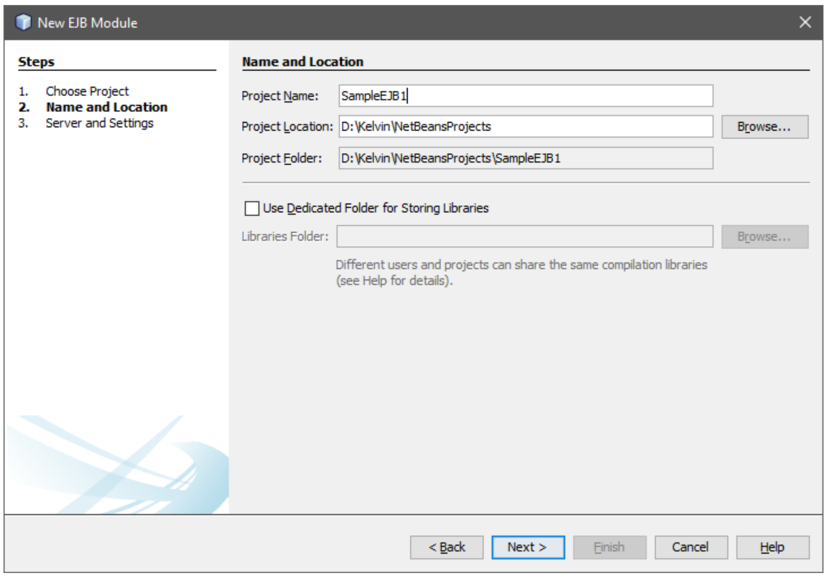
In NetBeans IDE, select **File > New Project**, or use this key combination *Ctrl + Shift + N*. You will see the following screen. Then, select **Java EE** in **Categories**, and **EJB Module** in **Projects**.



Creates a new Enterprise JavaBean (EJB) module in a standard IDE project. Standard projects use **an IDE-generated Ant build script** to build and run your project.

Figure 1 Create new project

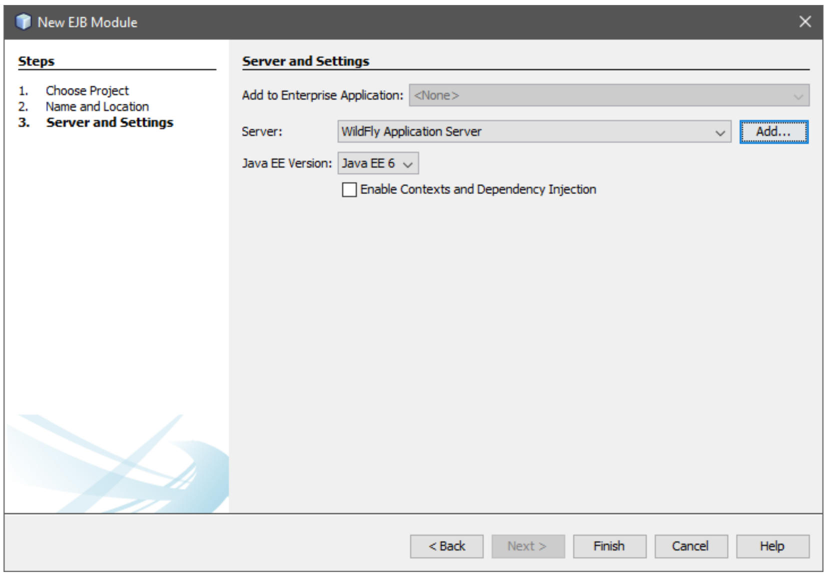
From **Figure 1**, click **Next** button, then you need to specify the **Project Name** and **Project Location**. For example, we use ***SampleEJB1***.



Project Name: SampleEJB1
Project Location: D:\Kelvin\NetBeansProjects
Project Folder: D:\Kelvin\NetBeansProjects\SampleEJB1
Use Dedicated Folder for Storing Libraries
Libraries Folder:
Different users and projects can share the same compilation libraries (see Help for details).

Figure 2 Create new project

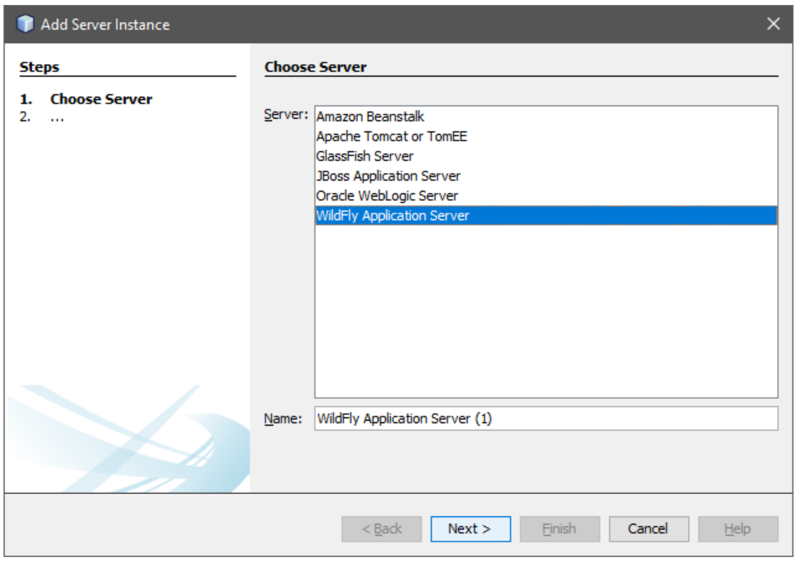
Then, as in **Figure 2**, click **Next** button to configure the **Server** settings. In this course, we recommend using the **WildFly Application Server** and **Java EE 6**.



Add to Enterprise Application: <None>
Server: WildFly Application Server
Java EE Version: Java EE 6
Enable Contexts and Dependency Injection

Figure 3 Configuring server settings

Note that, in case the **WildFly Application Server** is not presented, you click **Add…** button to open **Add Server Instance** window, as in **Figure 4**, **Figure 5**, and **Figure 6**. Then, choose **WildFly Application Server** in **Server** options, click **Next**, locate the directory of ***JBoss EAP*** in **Server Location** and **Server Configuration**. To finish the setup, click **Next** and configure the server's properties, finally, click **Finish**.


Add Server Instance

Steps

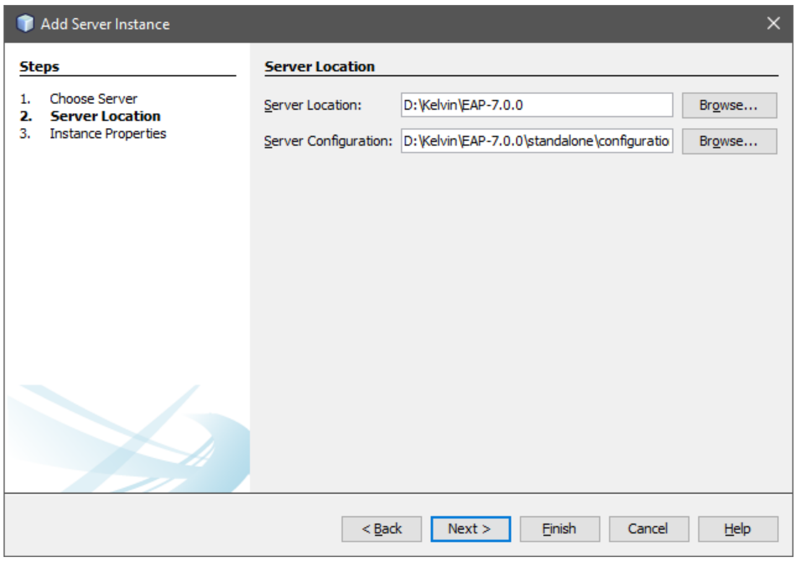
1. **Choose Server**
2. ...

Choose Server

Server:
- Amazon Beanstalk
- Apache Tomcat or TomEE
- GlassFish Server
- JBoss Application Server
- Oracle WebLogic Server
- WildFly Application Server

Name: WildFly Application Server (1)

Figure 4 Add Server Instance


Add Server Instance

Steps

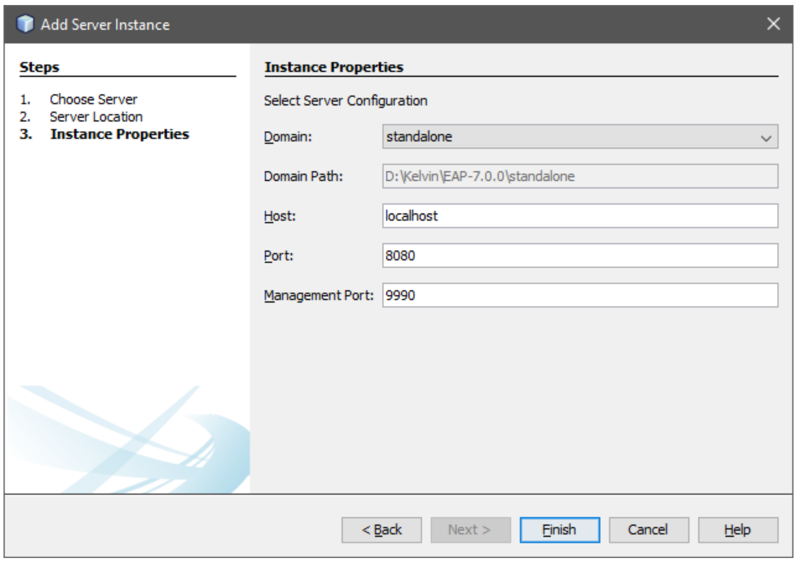
1. Choose Server
2. **Server Location**
3. Instance Properties

Server Location

Server Location: D:\Kelvin\EAP-7.0.0

Server Configuration: D:\Kelvin\EAP-7.0.0\standalone\configuratio

Figure 5 Browse the Server Location and Configuration directory


Add Server Instance

Steps

1. Choose Server
2. Server Location
3. **Instance Properties**

Instance Properties

Select Server Configuration

Domain: standalone

Domain Path: D:\Kelvin\EAP-7.0.0\standalone

Host: localhost

Port: 8080

Management Port: 9990

Figure 6 Configure the server's properties

Now, after completing **Figure 7**, we will see the following project created by NetBeans.

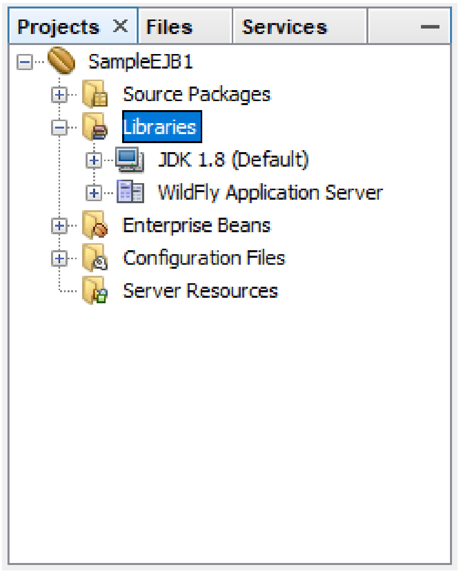


Figure 7 Project structure

2. Create a sample EJB

To create a simple EJB, we will use NetBeans "New" wizard. In the example given below, we will create a stateless EJB class named ***LibrarySessionBean*** under **SampleEJB1** project.

Select project ***SampleEJB1*** in project explorer window and right click on it. Select, **New > Session Bean**, you will see the **New Session Bean** wizard.

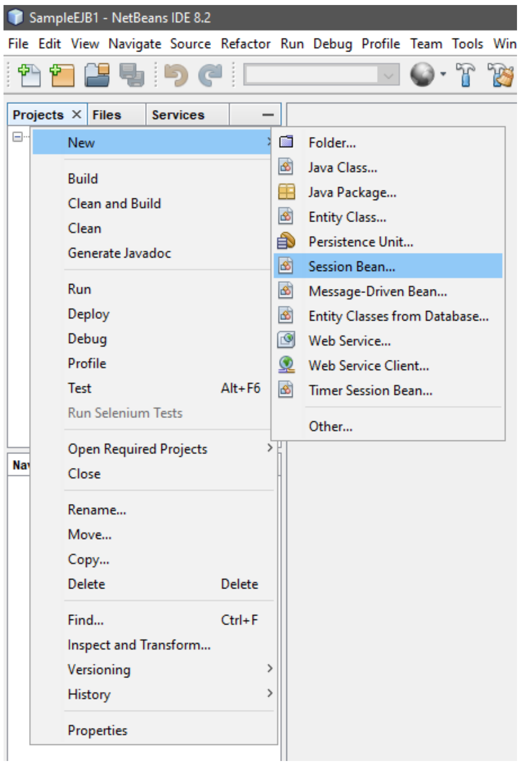
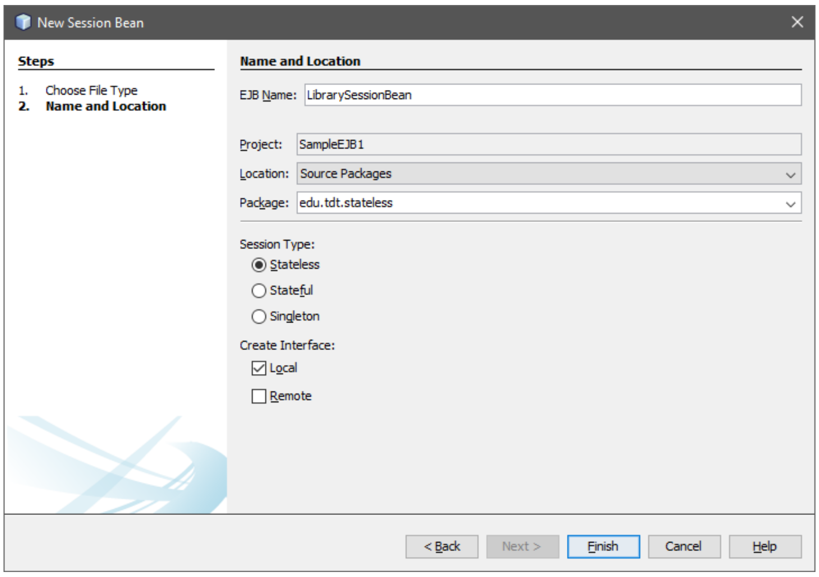


Figure 8 Create Session Bean

In the **New Session Bean** window, as in **Figure 9**, enter *EJB name* and *package name*, then, click **Finish** button. In this example, our bean name is ***LibrarySessionBean*** and package name is ***edu.tdt.stateless***. You should notice the **Session Type** and **Create Interface** options.



EJB Name: LibrarySessionBean
Project: SampleEJB1
Location: Source Packages
Package: edu.tdt.stateless
Session Type:
● Stateless
○ Stateful
○ Singleton
Create Interface:
☑ Local
☐ Remote

Figure 9 New Session Bean window

After completing the creation process, NetBeans create a package and two Java files, as in **Figure 10**, which are:

- ***LibrarySessionBean.java***: stateless session bean.
- ***LibrarySessionBeanLocal.java***: local interface for session bean.

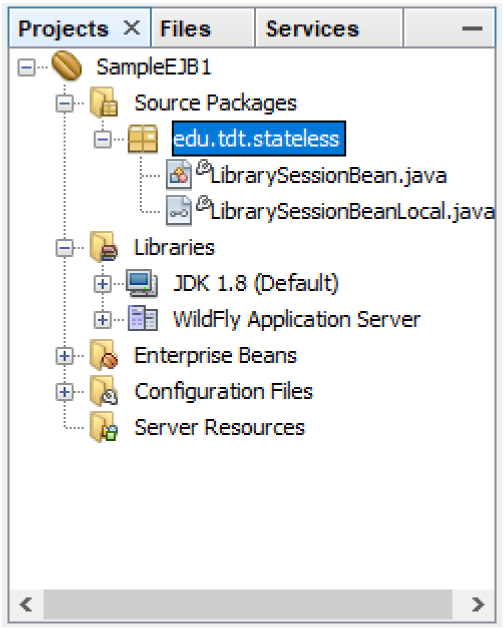


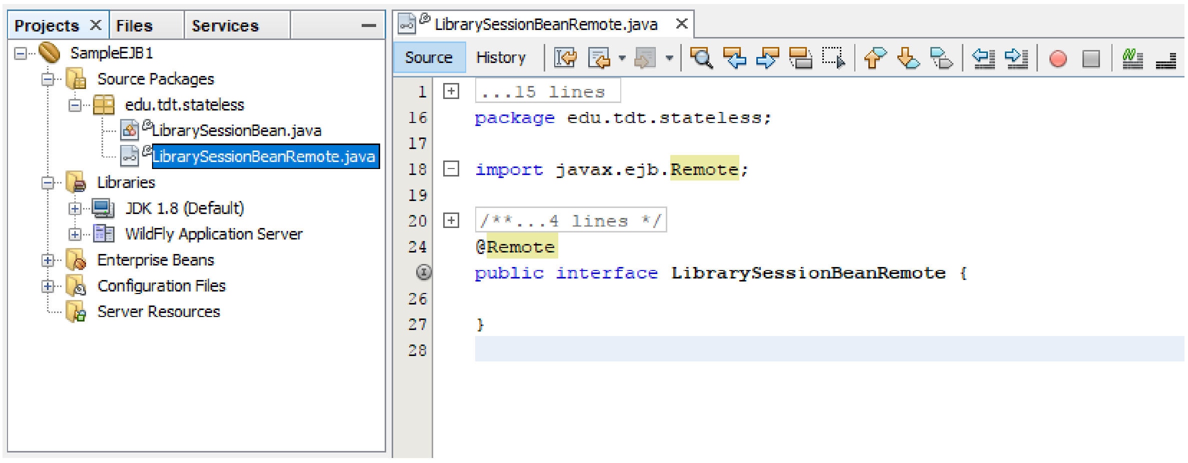
Figure 10 Project structure

Because we are going to access our EJB in a console based application, we change the ***local*** interface to ***remote*** interface. Remote/Local interface is used to expose business methods that an EJB must implement.

LibrarySessionBeanLocal is renamed to ***LibrarySessionBeanRemote*** and ***LibrarySessionBean*** implements ***LibrarySessionBeanRemote*** interface.

Besides that, we must change the import statement in our new ***LibrarySessionBeanRemote***, from **javax.ejb.Local** to **javax.ejb.Remote**, and the annotation from **@Local** to **@Remote**.

Figure 11 is what we get after editing the project.



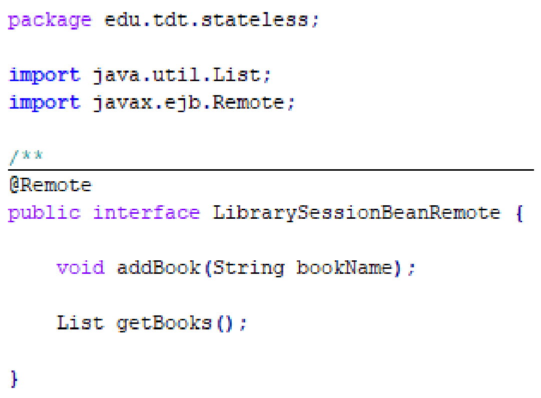
```
...15 lines
package edu.tdt.stateless;

import javax.ejb.Remote;

/**...4 lines */
@Remote
public interface LibrarySessionBeanRemote {

}
```

Figure 11 Editing the project

Let's implement the two Java files, as in **Figure 12** and **Figure 13**.

```
package edu.tdt.stateless;

import java.util.*;
import javax.ejb.Stateless;

@Stateless
public class LibrarySessionBean implements LibrarySessionBeanRemote {

    private List<String> bookShelf;

    public LibrarySessionBean()
    {
        bookShelf = new ArrayList<String>();
    }

    @Override
    public void addBook(String bookName)
    {
        this.bookShelf.add(bookName);
    }

    @Override
    public List getBooks()
    {
        return this.bookShelf;
    }

}
```

Figure 12 LibrarySessionBean.java



```
package edu.tdt.stateless;

import java.util.List;
import javax.ejb.Remote;

/**
@Remote
public interface LibrarySessionBeanRemote {

    void addBook(String bookName);

    List getBooks();

}
```

Figure 13 LibrarySessionBeanRemote.java

3. Build the project

To build the project to ***.class*** files, do the steps follows:

- Select ***SampleEJB1*** project in **Project Explorer** window.
- Right click on it to open context menu.
- Select **Clean and build**.

If everything is fine, you will see the following output in NetBeans consoles output.

```
ant -f D:\\Kelvin\\NetBeansProjects\\SampleEJB1 -Dnb.internal.action.name=rebuild clean dist
init:
undeploy-clean:
deps-clean:
clean:
init:
deps-jar:
Created dir: D:\Kelvin\NetBeansProjects\SampleEJB1\build\classes
Copying 2 files to D:\Kelvin\NetBeansProjects\SampleEJB1\build\classes\META-INF
Created dir: D:\Kelvin\NetBeansProjects\SampleEJB1\build\empty
Created dir: D:\Kelvin\NetBeansProjects\SampleEJB1\build\generated-sources\ap-source-output
Compiling 2 source files to D:\Kelvin\NetBeansProjects\SampleEJB1\build\classes
compile:
library-inclusion-in-archive:
Created dir: D:\Kelvin\NetBeansProjects\SampleEJB1\dist
Building jar: D:\Kelvin\NetBeansProjects\SampleEJB1\dist\SampleEJB1.jar
dist:
BUILD SUCCESSFUL (total time: 2 seconds)
```

4. Start the Application Server

To start the application server, do the following steps, as in **Figure 14**:

- In **Project Explorer** window, select **Services** tab.
- Expand **Server** section.
- Right click on **WildFly Application Server**, and select **Start**.

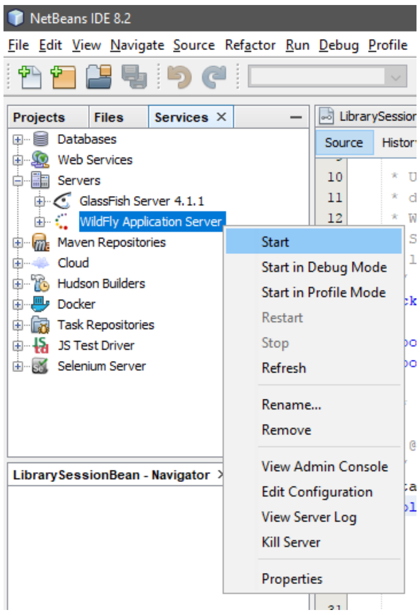


Figure 14 Start application server

If everything is fine, you will see the following output in NetBeans consoles output:

```
Calling "D:\Kelvin\EAP-7.0.0\bin\standalone.conf.bat"
…
…
…
20:11:39,188 INFO  [org.jboss.as] (Controller Boot Thread) WFLYSRV0025: JBoss EAP 7.0.0.GA
(WildFly Core 2.1.2.Final-redhat-1) started in 13047ms
```

5. Deploy the Project

To deploy the project, do the following steps, as in **Figure 15**:

- Change to **Projects** tab, select ***SampleEJB1*** project in **Project Explorer** window.
- Right click on it to open context menu.
- Select **Deploy**.

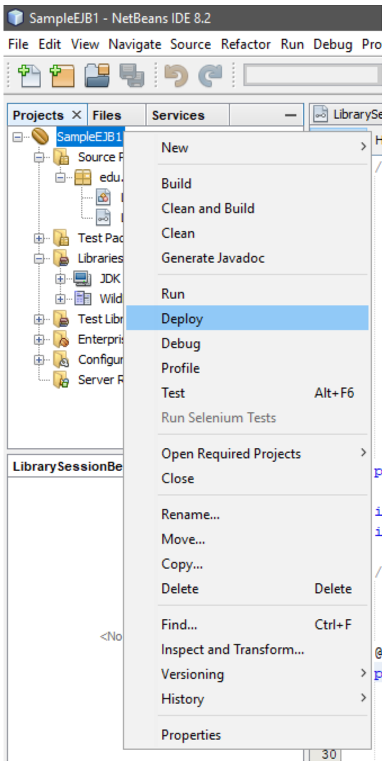


Figure 15 Deploy project

If everything is fine, you will see the following output in NetBeans consoles output:

```
Initial deploying SampleEJB1 to D:\Kelvin\EAP-7.0.0\standalone\deployments\SampleEJB1.jar
Completed initial distribution of SampleEJB1
Deploying D:\Kelvin\EAP-7.0.0\standalone\deployments\SampleEJB1.jar
Application Deployed
post-run-deploy:
run-deploy:
run:
BUILD SUCCESSFUL (total time: 3 seconds)
```

And, the following output in **WildFly Application Server Log Output**:

```
20:44:11,726 INFO  [org.jboss.weld.deployer] (MSC service thread 1-8) WFLYWELD0009: Starting
weld service for deployment SampleEJB1.jar
20:44:12,016  INFO   [org.jboss.as.clustering.infinispan]  (ServerService  Thread  Pool  --  74)
WFLYCLINF0002: Started client-mappings cache from ejb container
20:44:12,547 INFO  [org.jboss.as.server] (DeploymentScanner-threads - 1) WFLYSRV0010: Deployed
"SampleEJB1.jar" (runtime-name : "SampleEJB1.jar")
```

6. Create Client to access EJB

When you get to this step without any error, you can now create a client to access the EJB by doing the following steps in NetBeans:

- In NetBeans, select **File > New Project**.
- Select project type as *Java Application*. Click **Next** button
- Enter project name and location. Click **Finish** button. For example, ***SampleEJB1Tester***.
- Right click on project name in **Project Explorer** window, then select Properties.
- In **Project Properties** window, select **Libraries** option in **Categories**.

- In **Compile** tab, click **Add Project**, then select the EJB server project, for example, ***SampleEJB1***.
- Finally, add JBoss libraries using **Add JAR/Folder** button in **Compile** tab. JBoss libraries can be located at JBoss installation folder.

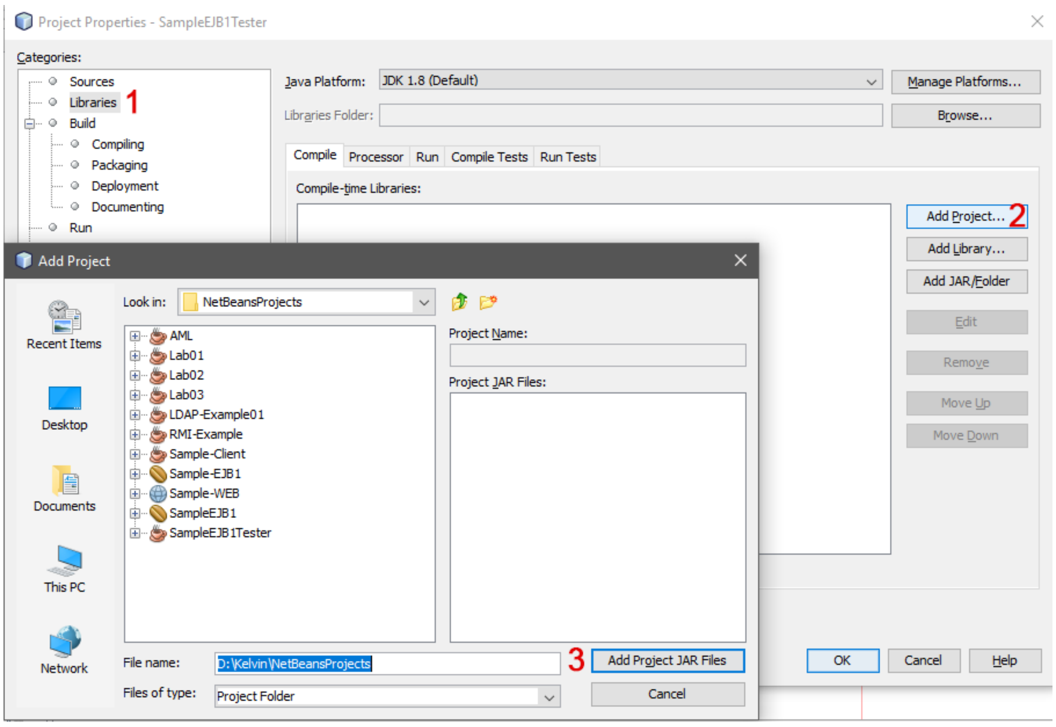


Figure 16 Configure project properties of Client (Part 1)

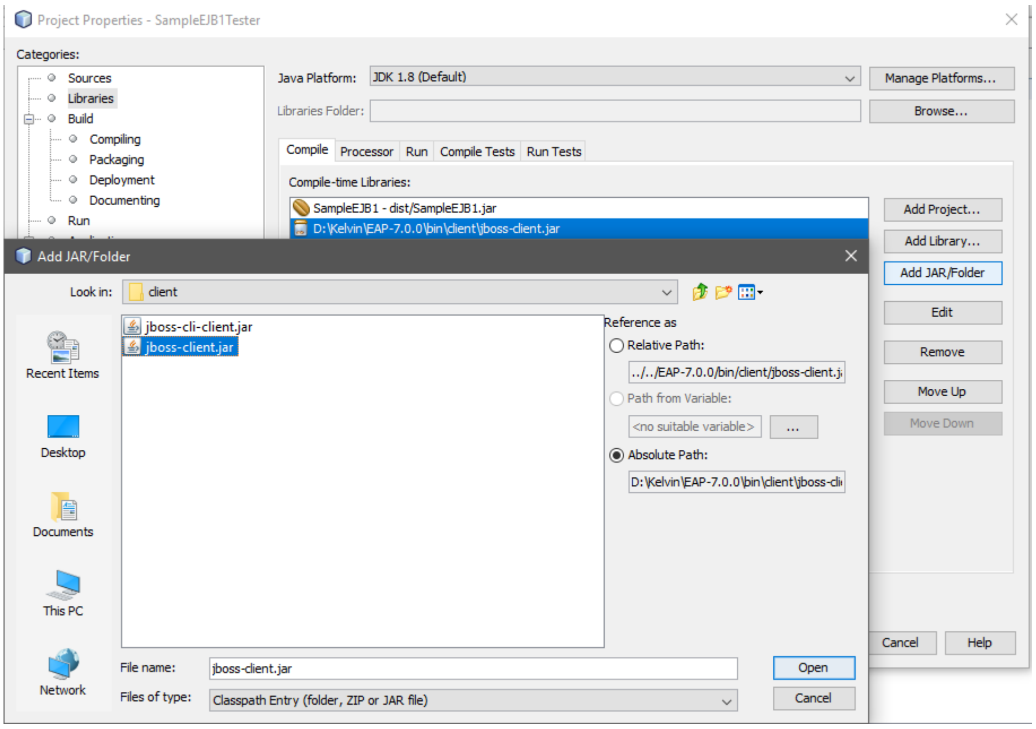
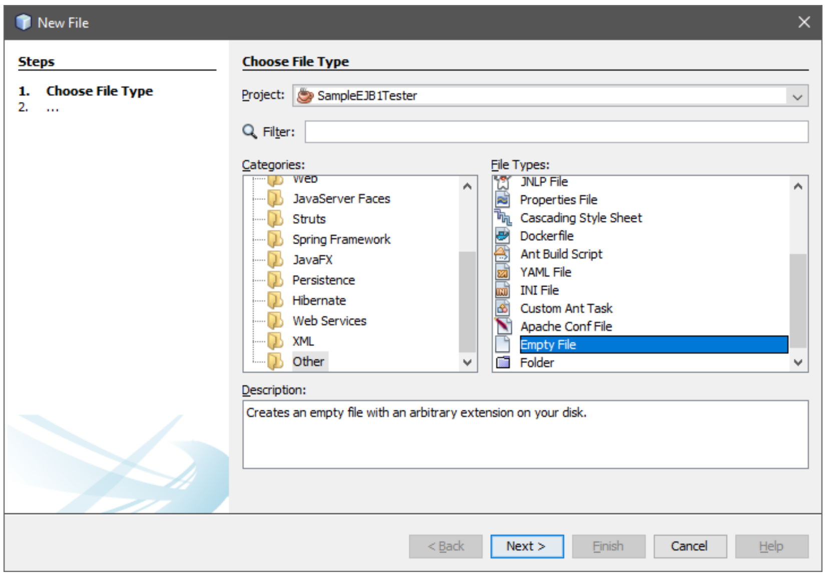


Figure 17 Configure project properties of Client (Part 2)

7. Create *jndi.properties*

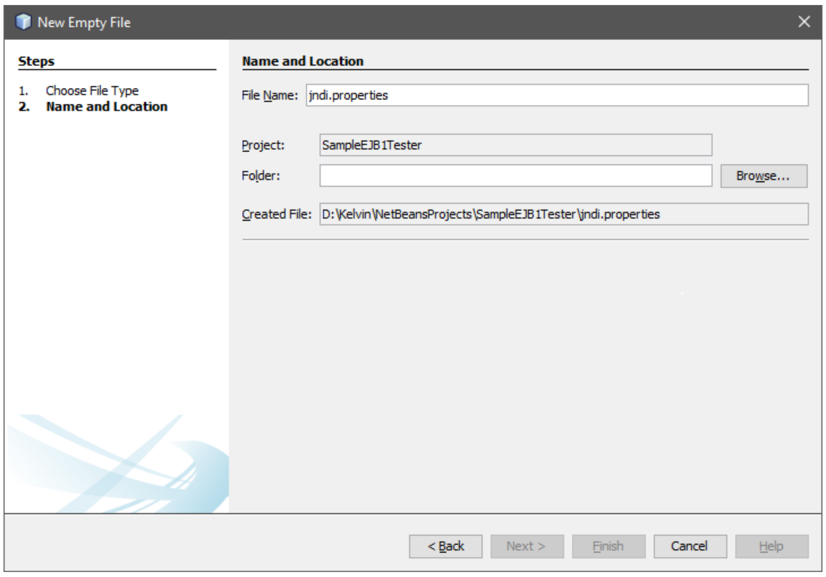
To specify environment properties, we need to define all of them in JNDI files, named by **jndi.properties**[1], for our example, do the following steps to create one:

- Right click on the client project, for example, ***SampleEJB1Tester***. Then, select **New > Other…**.
- Select **Other** in **Categories** section and **Empty File** in **File Types** section, click **Next**.
- Specify the **File Name** is ***jndi.properties***, for location, the default is root directory of project.



Creates an empty file with an arbitrary extension on your disk.

Figure 18 Create jndi.properties (Step 1)



File Name: jndi.properties
Project: SampleEJB1Tester
Folder:
Created File: D:\Kelvin\NetBeansProjects\SampleEJB1Tester\jndi.properties

Figure 19 Create jndi.properties (Step 2)

Now, in the editor window of ***jndi.properties***, paste the following code and save it.

```
jboss.naming.client.ejb.context=true
java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory
java.naming.factory.url.pkgs=org.jboss.ejb.client.naming
```

[1] http://docs.oracle.com/javase/jndi/tutorial/beyond/env/source.html

```
java.naming.provider.url=http-remoting://localhost:8080
java.naming.security.principal=admin
java.naming.security.credentials=12345678
```

8. Create *jboss-ejb-client.properties*

Since we use the JBoss EAP, we need to define the ***jboss-ejb-client.properties***, including the current configuration of JBoss EAP server. Do the following steps (like **Section 7**):

- In the client project, right click on **src** folder. Then, select **New** > **Other…**.
- Select **Other** in **Categories** section and **Empty File** in **File Types** section, click **Next**.
- Specify the **File Name** is ***jboss-ejb-client.properties***.

Paste the following lines to ***jboss-ejb-client.properties*** file:

```
remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
remote.connections=default
remote.connection.default.host=localhost
remote.connection.default.port=8080
remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=false
remote.connection.default.username=admin
remote.connection.default.password=12345678
```

(*) The very important thing is the ***jboss-ejb-client.properties*** must be in **src** folder.

9. Implement the *EJBTester.java*

In this section, we will focus on ***Client*** side and implement the ***EJBTester.java*** class. To begin, we need to create a package ***edu.tdt.test*** and ***EJBTester.java*** under it.

```
package edu.tdt.test;

import edu.tdt.stateless.LibrarySessionBean;
import edu.tdt.stateless.LibrarySessionBeanRemote;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.List;
import java.util.Properties;
import java.util.Scanner;
import javax.naming.InitialContext;
import javax.naming.NamingException;


public class EJBTester {

    private Properties props;
    private InitialContext ctx;

    public EJBTester()
    {
        readJNDI();
    }

    /**
     * Read the JNDI properties file
     */
    private void readJNDI()
    {
        props = new Properties();

        try
        {
            props.load(new FileInputStream("jndi.properties"));
        } catch(IOException e)
        {
            System.err.println(e.toString());
```

```java
        }

        try
        {
            ctx = new InitialContext(props);
            //ctx.close();
        } catch (NamingException ex)
        {
            System.err.println(ex.toString());
        }
    }

    private String getJNDI()
    {
        String appName = "";
        String moduleName = "SampleEJB1";
        String distinctName = "";
        String sessionBeanName = LibrarySessionBean.class.getSimpleName();
        String viewClassName = LibrarySessionBeanRemote.class.getName();

        return "ejb:"+appName+"/"+moduleName+"/"+distinctName+"/"+sessionBeanName+"!"+viewClassName;
    }

    /**
     * Show the GUI in console window
     */
    private void showGUI()
    {
        System.out.println("\n===============================");
        System.out.println("Welcome to TDTU Bookstore");
        System.out.println("===============================");
        System.out.print("Options: \n1. Add Book \n2. List All Books \n3. Exit \nEnter Choice: ");
    }

    /**
     * Declare a bean to invoke getBooks() method in LibrarySessionBeanRemote
     */
    private void getAllBooks()
    {
        try
        {
            // We can define another bean to access the LibrarySessionBeanRemote
            LibrarySessionBeanRemote libBean2 = (LibrarySessionBeanRemote) ctx.lookup(getJNDI());
            List<String> booksList = libBean2.getBooks();

            // Print all books
            System.out.println("\n===============================");
            System.out.println("Listing Books in TDTU Bookstore");
            System.out.println("===============================");
            for (int i = 0; i < booksList.size(); i++)
            {
                System.out.println(i + "\t" + booksList.get(i));
            }
            System.out.println("");

        } catch (NamingException ex)
        {
            System.err.println(ex.toString());
        }
    }

    /**
     * Test the Stateless EJB
     */
    public void testStatelessEJB()
    {
        try
        {
            // Scanner definition
            Scanner sc = new Scanner(System.in);

            // Lookup the LibrarySessionBeanRemote
            LibrarySessionBeanRemote libBean = (LibrarySessionBeanRemote) ctx.lookup(getJNDI());

            int choice = 0;
            while(choice != 3)
            {
                this.showGUI();

                // Use this approach because nextInt will cause error to nextLine()
                choice = Integer.parseInt(sc.nextLine());

                if(choice == 1)
                {
                    // Add a book
                    System.out.print("Enter book name: ");
                    String bookName = sc.nextLine();
                    libBean.addBook(bookName);
                }
                else if(choice == 2)
                {
                    // Print all books
                    getAllBooks();
                }
                else
                {
                    // Exit
                    break;
                }
            }

            sc.close();

        } catch (NamingException ex)
        {
            System.err.println(ex.toString());
        }
    }
}
```

Let's briefly introduce the ***EJBTester*** class:

- We have defined the JNDI properties file, thus we need to read it in ***EJBTester*** by calling method ***readJNDI()***. However, we also create ***jboss-ejb-client.properties***, but we don't need to explicitly read it in our code, just put it in **src** folder and JBoss will handle.
- In this lab, we don't need to construct a Web page, so working in console window is enough. Therefore, to show the GUI to user, we call ***showGUI()*** method.
- Before discussing ***getAllBooks()*** method, we should discuss ***testStatelessEJB()*** method. It will define a session bean to access the ***LibrarySessionBeanRemote*** (in **Section 2**) to invoke the defined public methods.
- The question here is how the EJB know exactly the location of called class in server. Therefore, we need to specify the address. The address in EJB has 5 parts: (1) *appName*, (2) *moduleName*, (3) *distinctName*, (4) *sessionBean*, and (5) *viewClassName*. We define all parameters in **getJNDI()** method.
- Now, ***getAllBooks()*** method, firstly, we need to notice that, in ***testStatelessEJB()*** method, we declare ***libBean*** to access the remote methods. In ***getAllBooks()*** method, we declare ***libBean2***, that means, we can declare many beans to access the remote methods in the same program.
- In ***testStatelessEJB()***, since the business logic in this scenario is only two functions, add new book and list all books, we build a choice list to interact with users. Notice that, all methods which process the bookstore's data must be defined in server side (as in **Section 2**).

10. Run Client to access EJB

To run the **Client**, we need to define the ***main*** method. Create a new Java class, named by ***Test.java***, and define the ***main()*** method as follows:

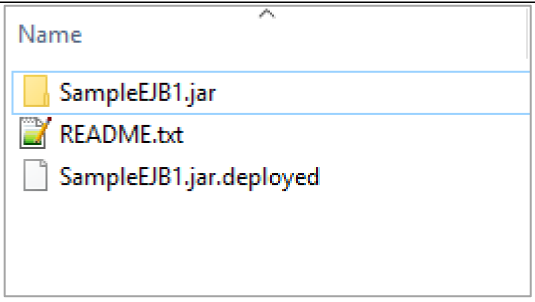
```
package edu.tdt.test;

public class Test {

    public static void main(String[] args)
    {
        EJBTester ejb = new EJBTester();
        ejb.testStatelessEJB();
    }

}
```

Until this step, everything becomes easy, just hit **Run Project** in NetBeans (or press the **F6** key).

IV. Why Error?

No.	***Error/Situation***	***Solution***
1	Don't start the server	The **WildFly Application Server** must be start before deploying and/or running the project.
2	Don't deploy the server project	Before running the client project, the server project must be deployed to **WildFly Application Server**.
3	Wrong username/password of **WildFly Application Server**	Change to the correct username/password.
4	No EJB receiver available for handling	There are many reasons for this error, you should check the above errors/situations. If none of them, you should check whether you call appropriate method. Other case, you should check the **%JBOSS_HOME%/standalone/deployments/** for the present of ***.jar.deployed** file[2]. This file indicates that the server project has been successfully deployed to **WildFly Application Server**.

[2] For all marker types, read *http://it.tdt.edu.vn/~dhphuc/teaching/is502052/files/marker_types.txt*

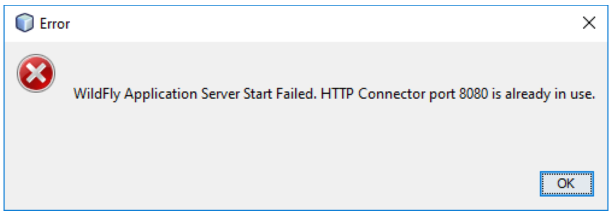
No.	Error/Situation	Solution
		 Name SampleEJB1.jar README.txt SampleEJB1.jar.deployed *Figure 20 Example marker file*

V. Exercises

1. Implementing the following methods on the server-side program:
   - Remove all books.
   - Remove a specific book, given a ***bookID***. To implement this method, you also need to add ***bookID*** property in ***LibrarySessionBean*** class.
   - Edit a ***bookName*** given ***bookID***.
   - Find a book given ***bookName*** or ***bookID***.
2. In the sample program, when we turn off the **WildFly Application Server**, all data in ***bookShelf*** will be lost. Considering this situation, how can you overcome it?

APPENDIX A: How to Fix the Port in Use Error

This appendix is applied when you get the dialogue that states the "WildFly Application Server Start Failed. HTTP Connector port 8080 is already in use.", as shows in **Figure 21**.



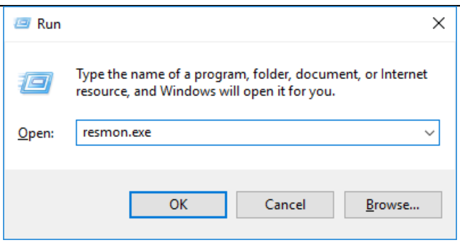
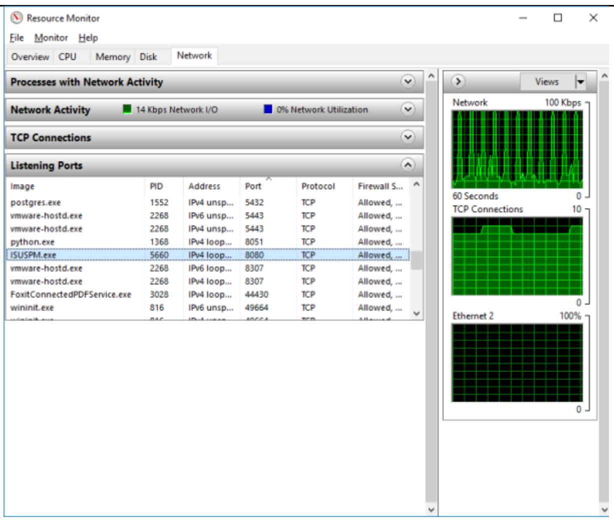
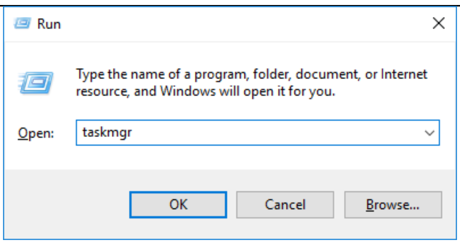
Error

WildFly Application Server Start Failed. HTTP Connector port 8080 is already in use.

OK

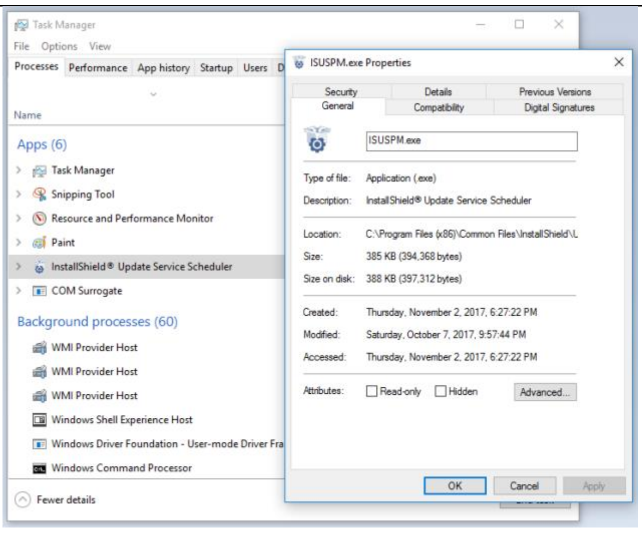
Figure 21 Port error dialogue

To solve this error, we perform the following steps (applied for Windows 7 and later):

Step 1: Open Run box and type resmon.exe, then press Enter.	 Run Type the name of a program, folder, document, or Internet resource, and Windows will open it for you. Open: resmon.exe OK Cancel Browse...
Step 2: In Resource Manager windows, switch to Network tab and expand Listening Ports section.	 Resource Monitor File Monitor Help Overview CPU Memory Disk Network Processes with Network Activity Network Activity 🟩 14 Kbps Network I/O 🟦 0% Network Utilization TCP Connections Listening Ports Image / PID / Address / Port / Protocol / Firewall S... postgres.exe / 1552 / IPv4 unsp... / 5432 / TCP / Allowed, ... vmware-hostd.exe / 2268 / IPv6 unsp... / 5443 / TCP / Allowed, ... vmware-hostd.exe / 2268 / IPv4 unsp... / 5443 / TCP / Allowed, ... python.exe / 1368 / IPv4 loop... / 8051 / TCP / Allowed, ... ISUSPM.exe / 5660 / IPv4 loop... / 8080 / TCP / Allowed, ... vmware-hostd.exe / 2268 / IPv6 loop... / 8307 / TCP / Allowed, ... vmware-hostd.exe / 2268 / IPv4 loop... / 8307 / TCP / Allowed, ... FoxitConnectedPDFService.exe / 3028 / IPv4 loop... / 44430 / TCP / Allowed, ... wininit.exe / 816 / IPv6 unsp... / 49664 / TCP / Allowed, ... Views Network 100 Kbps 60 Seconds 0 TCP Connections 10 0 Ethernet 2 100% 0
Step 3: Find the port that you want to find out whether it is in-listening or not. In this case, we focus on port 8080.	
Step 4: After figuring out the application is listening port 8080, you then open Task Manager, by enter "taskmgr" in Run windows.	 Run Type the name of a program, folder, document, or Internet resource, and Windows will open it for you. Open: taskmgr OK Cancel Browse...

Step 5:

Find the task that is currently listening port 8080, and then end it.



Task Manager — File Options View — Processes | Performance | App history | Startup | Users

Name

Apps (6)
- Task Manager
- Snipping Tool
- Resource and Performance Monitor
- Paint
- InstallShield® Update Service Scheduler
- COM Surrogate

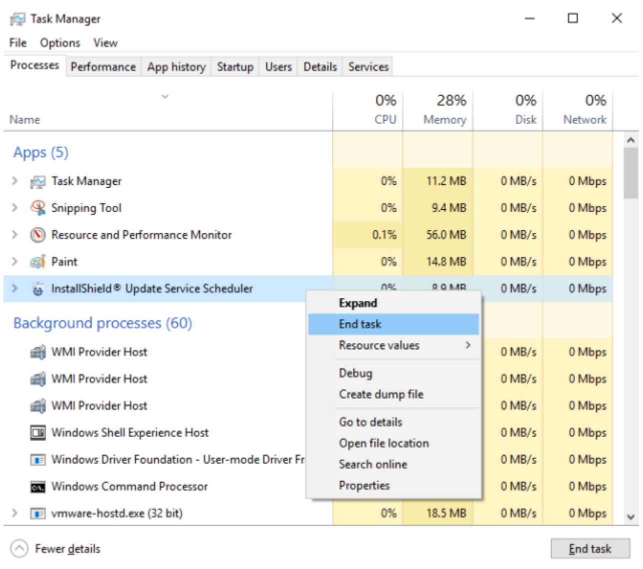
Background processes (60)
- WMI Provider Host
- WMI Provider Host
- WMI Provider Host
- Windows Shell Experience Host
- Windows Driver Foundation - User-mode Driver Fra
- Windows Command Processor

Fewer details

ISUSPM.exe Properties — Security | Details | Previous Versions | General | Compatibility | Digital Signatures

	ISUSPM.exe
Type of file:	Application (.exe)
Description:	InstallShield® Update Service Scheduler
Location:	C:\Program Files (x86)\Common Files\InstallShield\U
Size:	385 KB (394,368 bytes)
Size on disk:	388 KB (397,312 bytes)
Created:	Thursday, November 2, 2017, 6:27:22 PM
Modified:	Saturday, October 7, 2017, 9:57:44 PM
Accessed:	Thursday, November 2, 2017, 6:27:22 PM
Attributes:	☐ Read-only ☐ Hidden Advanced...

OK Cancel Apply



Task Manager — File Options View — Processes | Performance | App history | Startup | Users | Details | Services

Name	0% CPU	28% Memory	0% Disk	0% Network
Apps (5)				
Task Manager	0%	11.2 MB	0 MB/s	0 Mbps
Snipping Tool	0%	9.4 MB	0 MB/s	0 Mbps
Resource and Performance Monitor	0.1%	56.0 MB	0 MB/s	0 Mbps
Paint	0%	14.8 MB	0 MB/s	0 Mbps
InstallShield® Update Service Scheduler	0%	8.9 MB	0 MB/s	0 Mbps
Background processes (60)				
WMI Provider Host			0 MB/s	0 Mbps
WMI Provider Host			0 MB/s	0 Mbps
WMI Provider Host			0 MB/s	0 Mbps
Windows Shell Experience Host			0 MB/s	0 Mbps
Windows Driver Foundation - User-mode Driver Fr			0 MB/s	0 Mbps
Windows Command Processor			0 MB/s	0 Mbps
vmware-hostd.exe (32 bit)	0%	18.5 MB	0 MB/s	0 Mbps

(Notice that the task maybe different across computers.)