Integrating NWDS with a Non-SAP Server (JBoss AS) to Develop and Deploy Java EE Applications

Applies to:

This article applies to SAP NetWeaver Developer Studio, SAP NetWeaver 7.1 CE SP03 PAT0000

Summary

The procedure of integrating SAP NetWeaver Developer Studio with a non-SAP Server (JBoss AS) is presented. The steps involved in creating EJB, web, enterprise application, and application client projects and deploying them in JBoss Application Server are explained with appropriate screenshots. This would be useful in scenarios where existing or new J2EE applications need to be tested in non-SAP servers before being deployed in the SAP NetWeaver Application Server.

Author: Seema Thejraj

Company: HCL Technologies Limited

Created on: 23 August 2006

Author Bio

Seema Thejraj is working as an Associate Project Manager in Engineering and R & D services of HCL Technologies Ltd., Chennai.

Table of Contents

Introduction

This article discusses the steps that are required to develop and deploy an enterprise application using NWDS and JBoss Application Server using a sample Contacts Management scenario. Basic CRUD operations for contacts and some utility classes have been created in the EJB project. List contacts JSP sample has been developed in the Web Project. This sample is not a complete application and is meant only for demonstrating how Java EE applications can be developed and deployed to JBoss.

Setting up JBoss Server

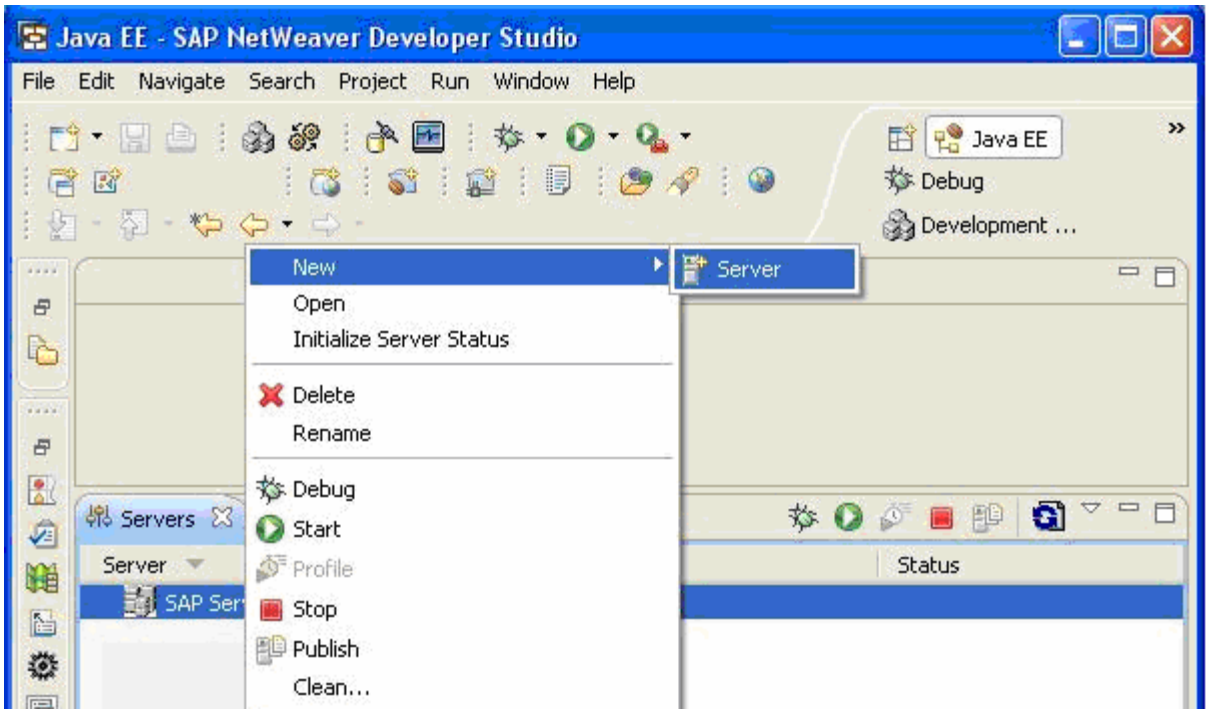
Prerequisites

JDK 1.6.x should be installed

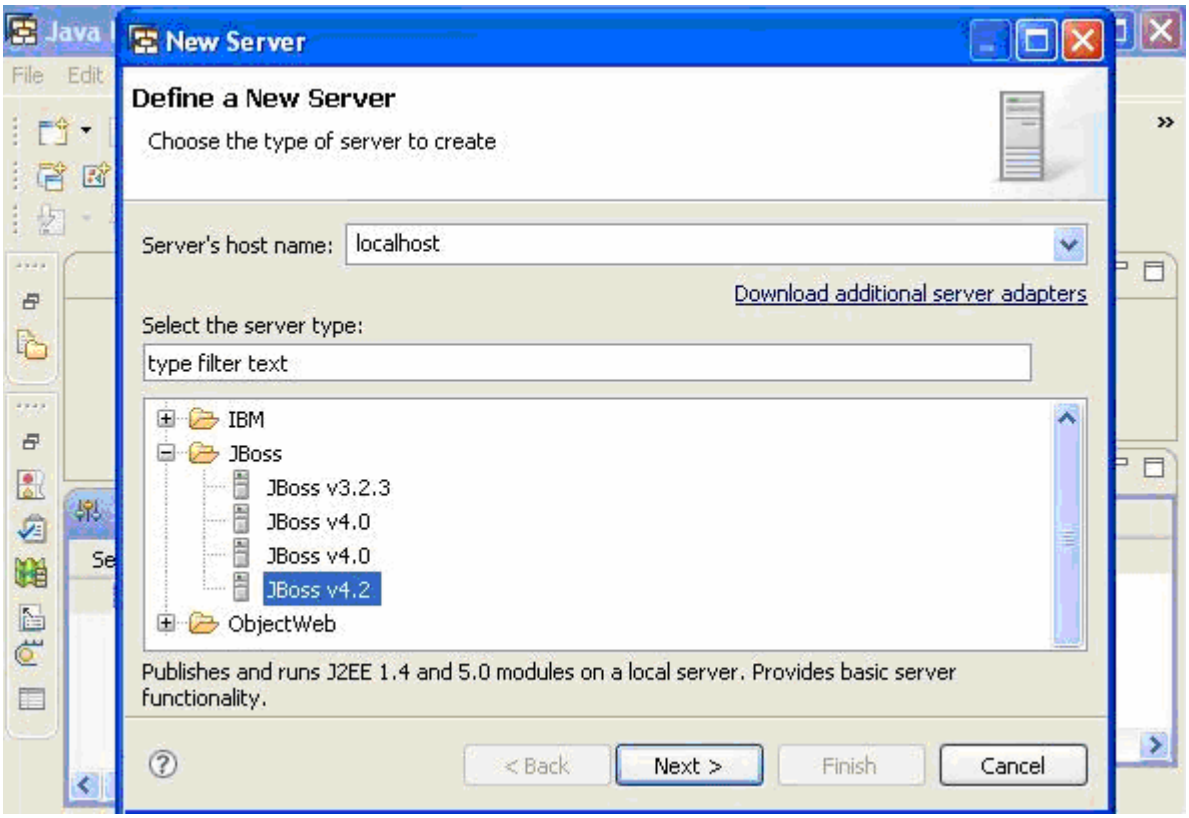
JBoss 4.2.x should be installed in local system.

Configuring the JBoss Server in NWDS

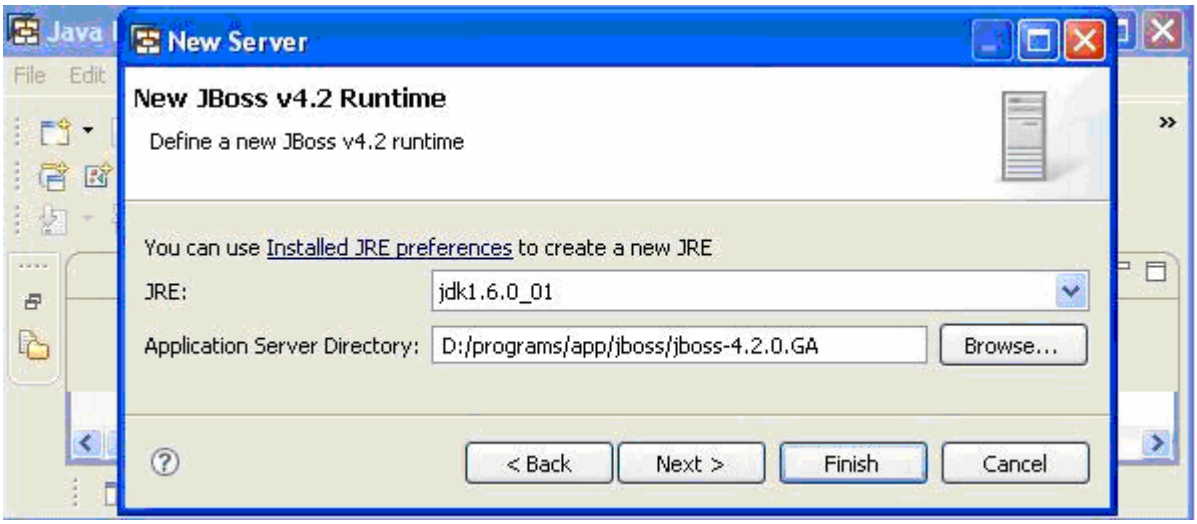
Open the SAP NetWeaver Developer Studio for CE 7.1. Switch to the Server view in the J2EE Perspective. Right click anywhere in the server view. Choose New > Server in the Context Menu that is displayed.



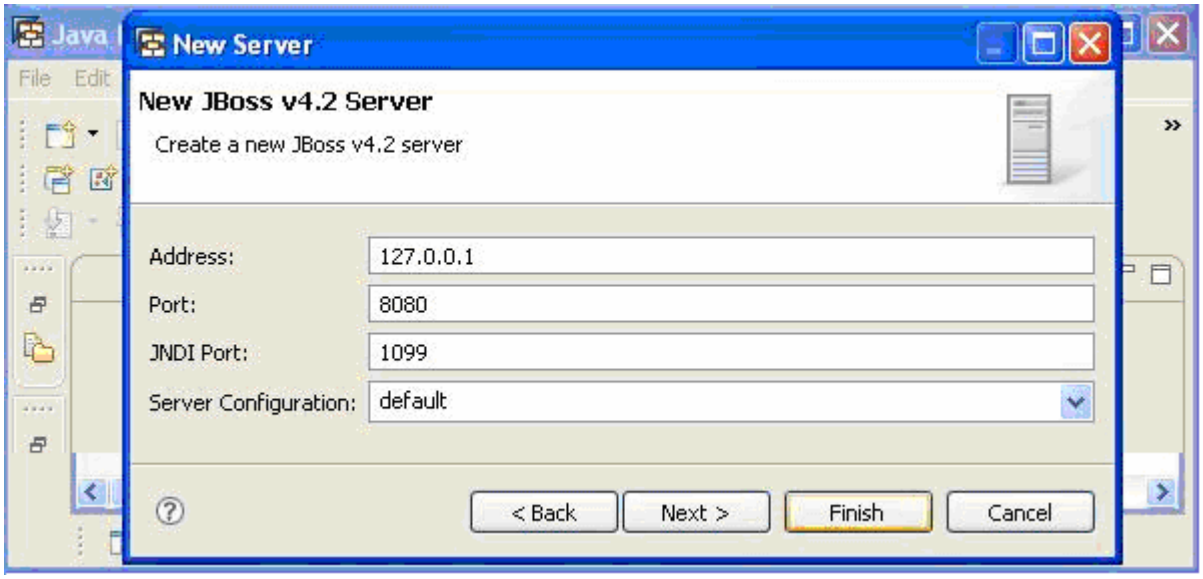
Leave the hostname as localhost as JBoss is installed in local system. (If any other hostname is specified only Basic and SAP nodes are shown. All other Non SAP servers are not listed.) Choose the Server type as JBoss v4.2. Click Next.



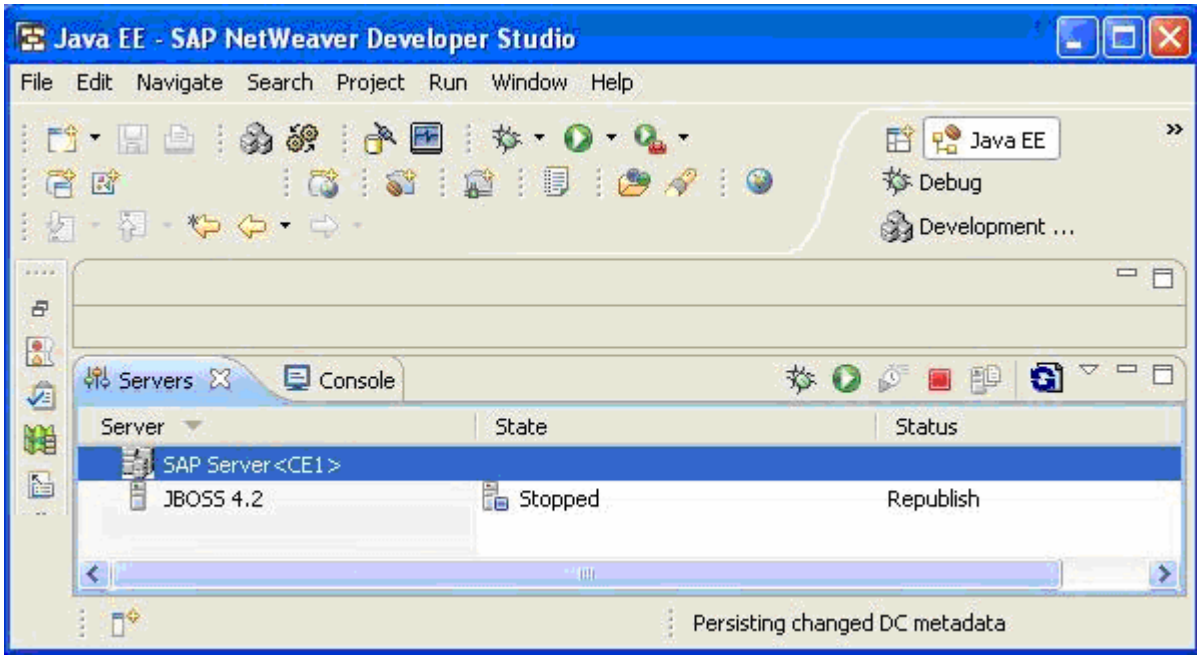
Choose jdk1.6.0_01 for JRE and the JBoss Home folder for the Application Server Directory. Click Next.



Leave the default values for the Server Address, Port, JNDI Port and Server Configuration. Click Finish.

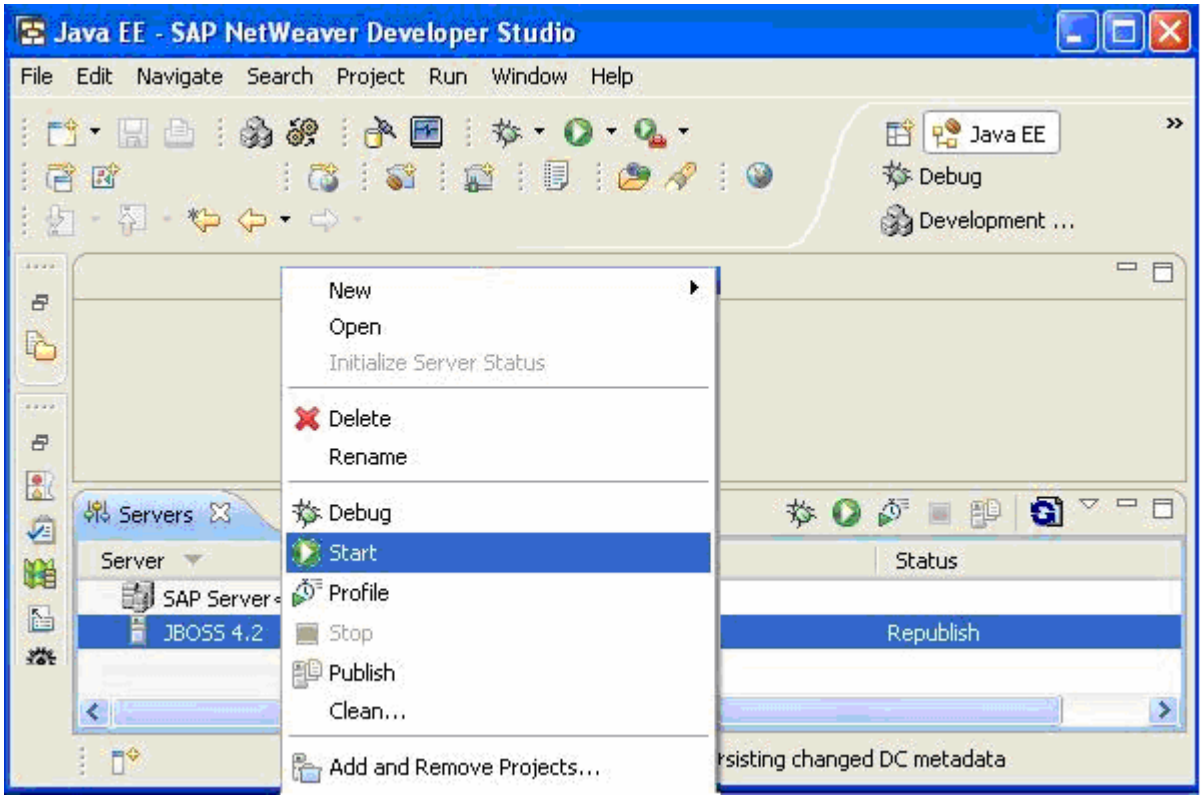


A JBoss4.2 node is shown in the servers view in the stopped state.

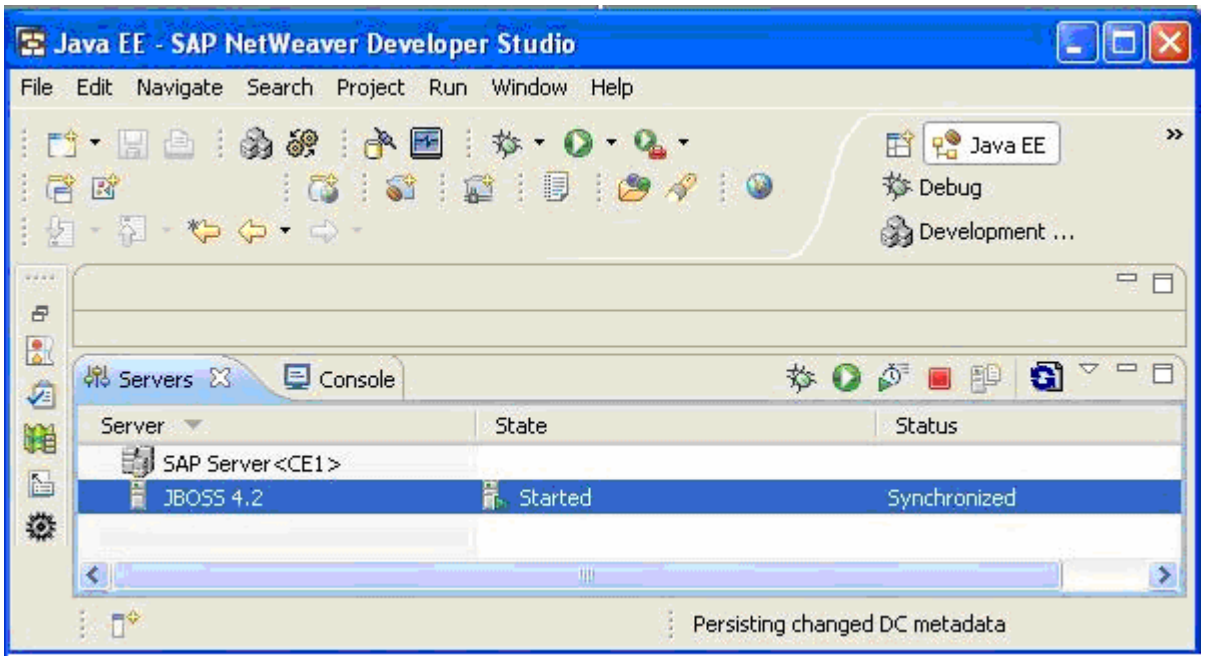


Starting the JBoss Server

Right click the JBoss server node and click Start.



If the JBoss server is started, the state is set to Started.

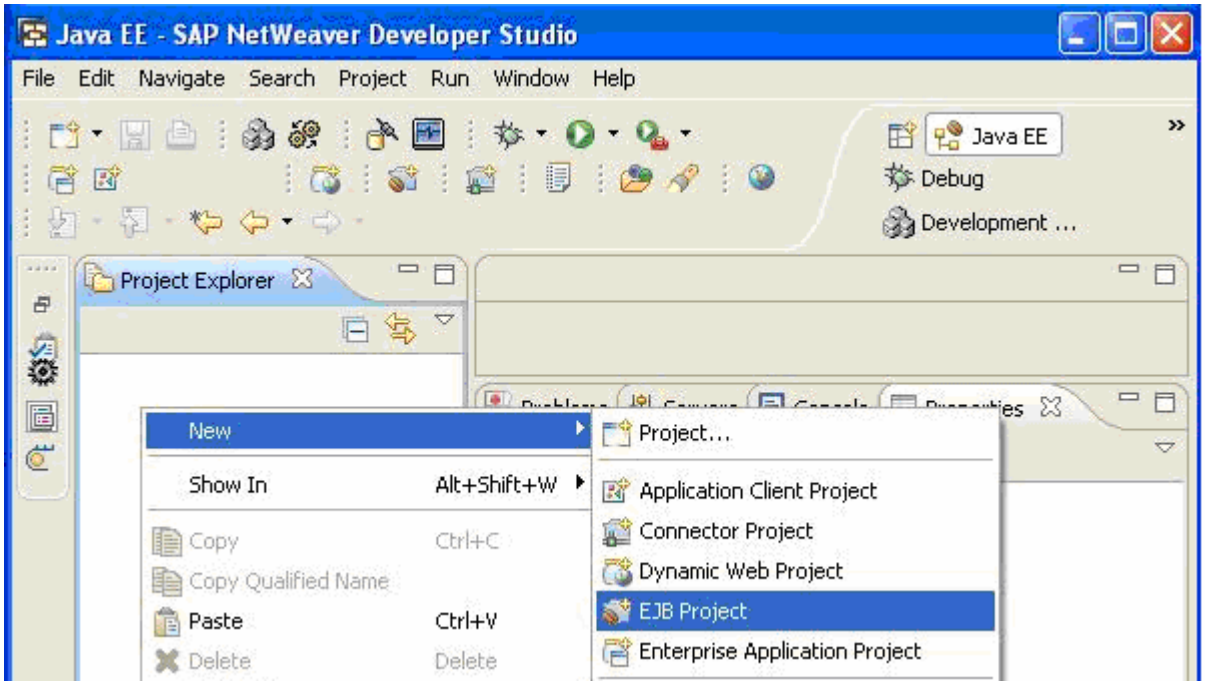


You can verify the server startup success by accessing the URL http://localhost:8080

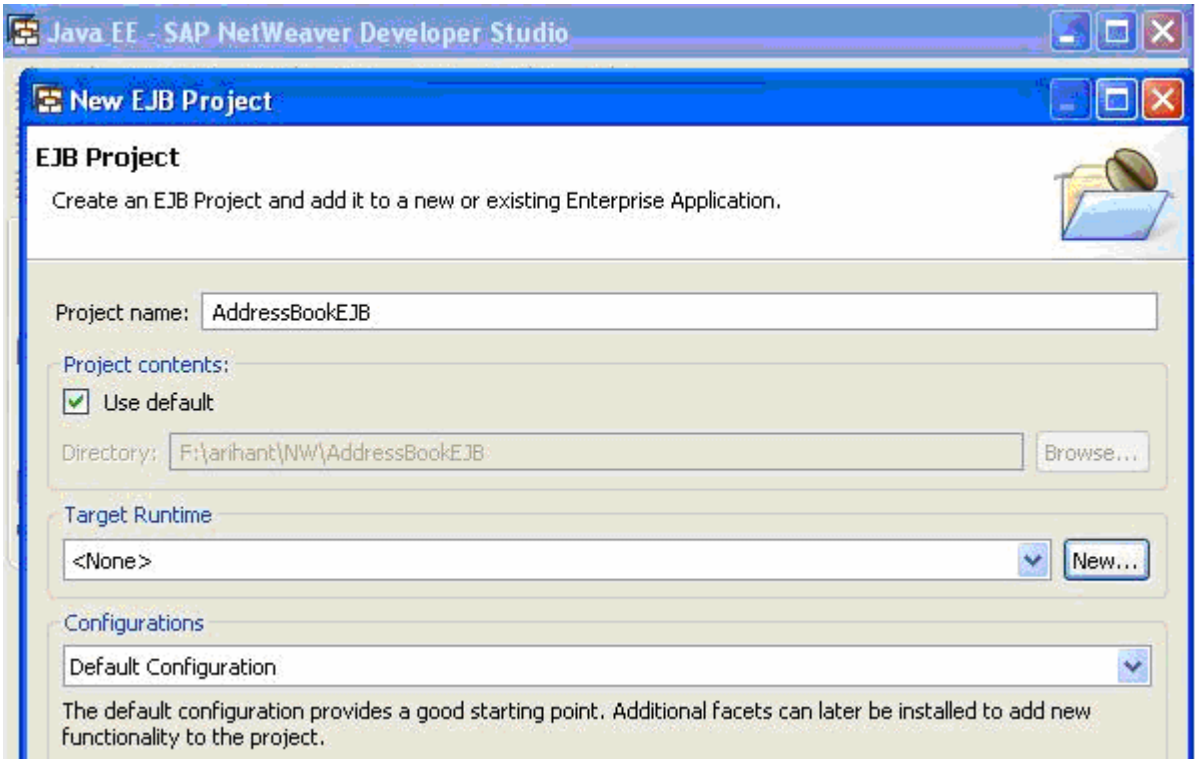
Developing EJB Component

Creating an EJB Project

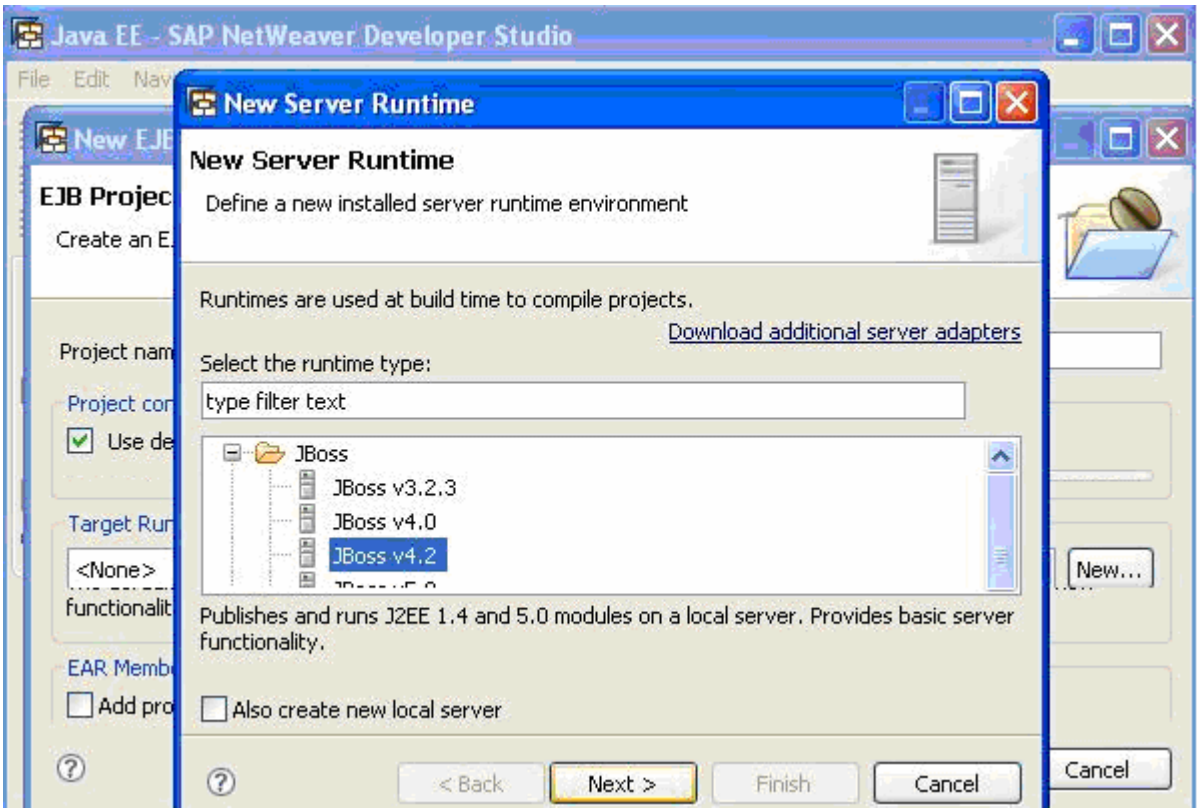
Right click on the Project Explorer view. Choose New > EJB Project in the context menu that is displayed.



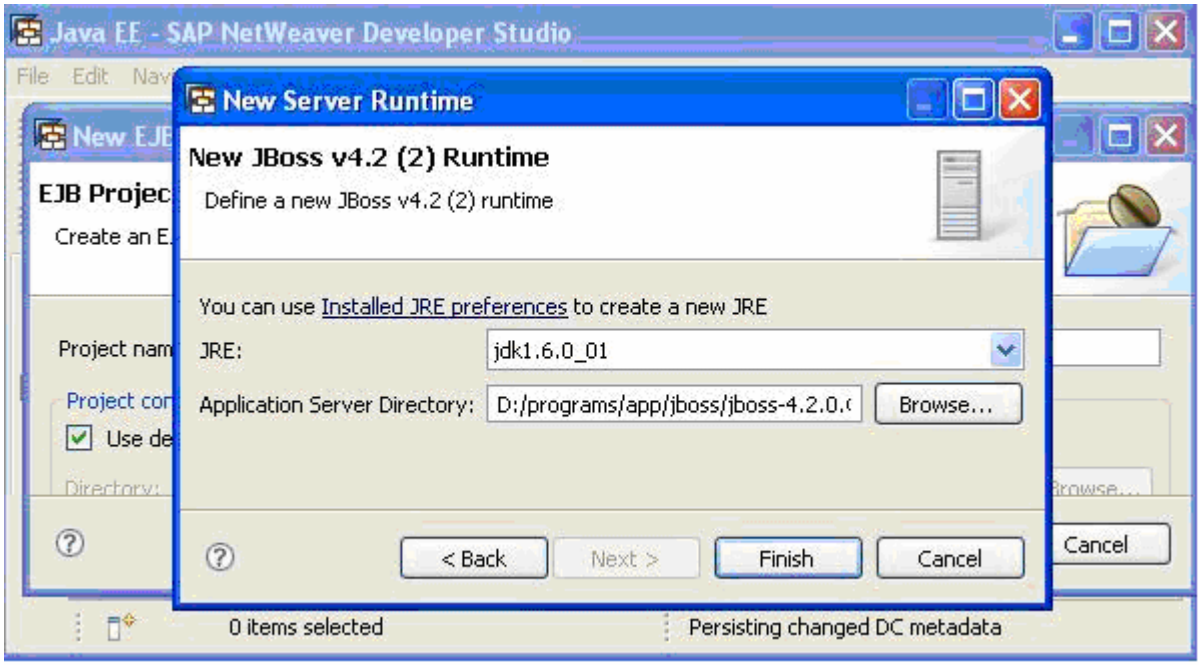
Enter the Project Name as AddressBookEJB.



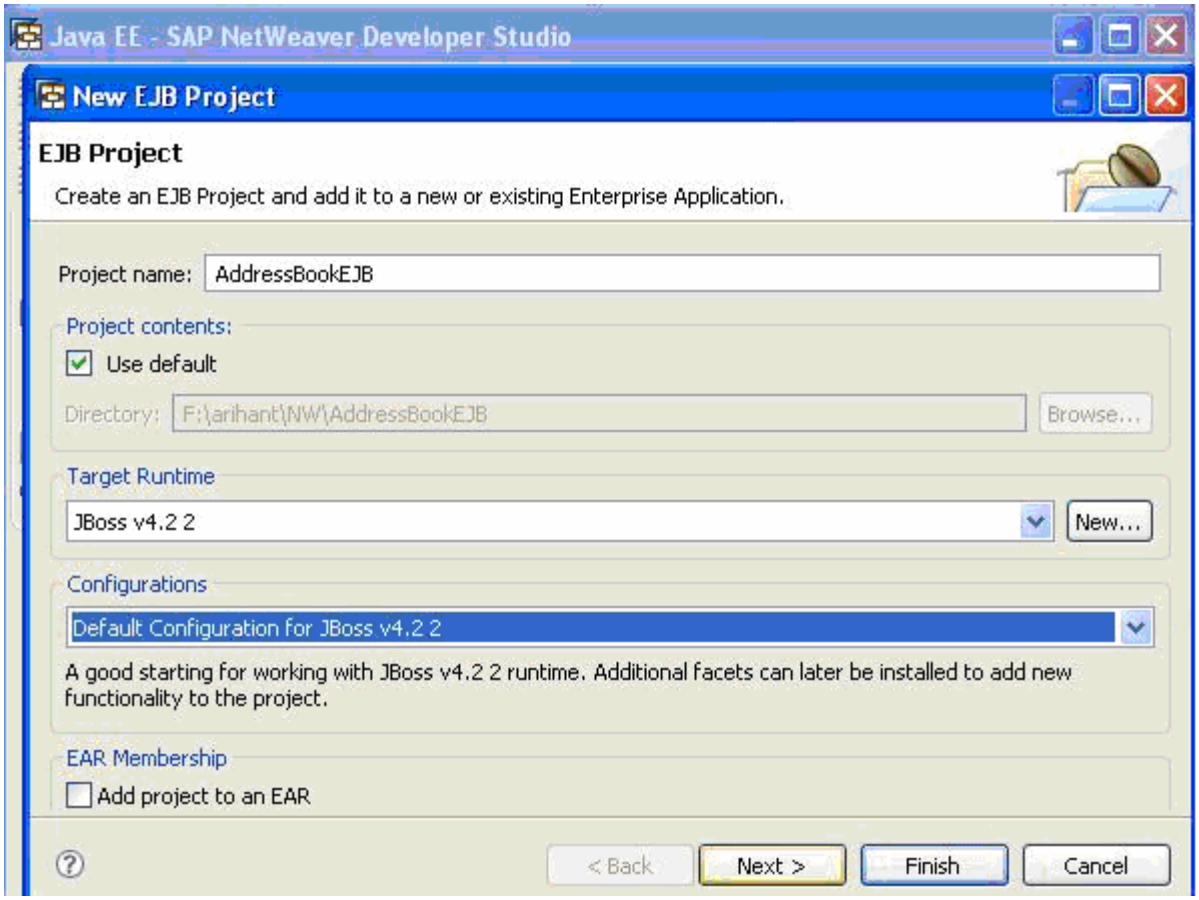
Click New for Target Runtime. Choose the runtime type as JBoss v4.2. Click Next.



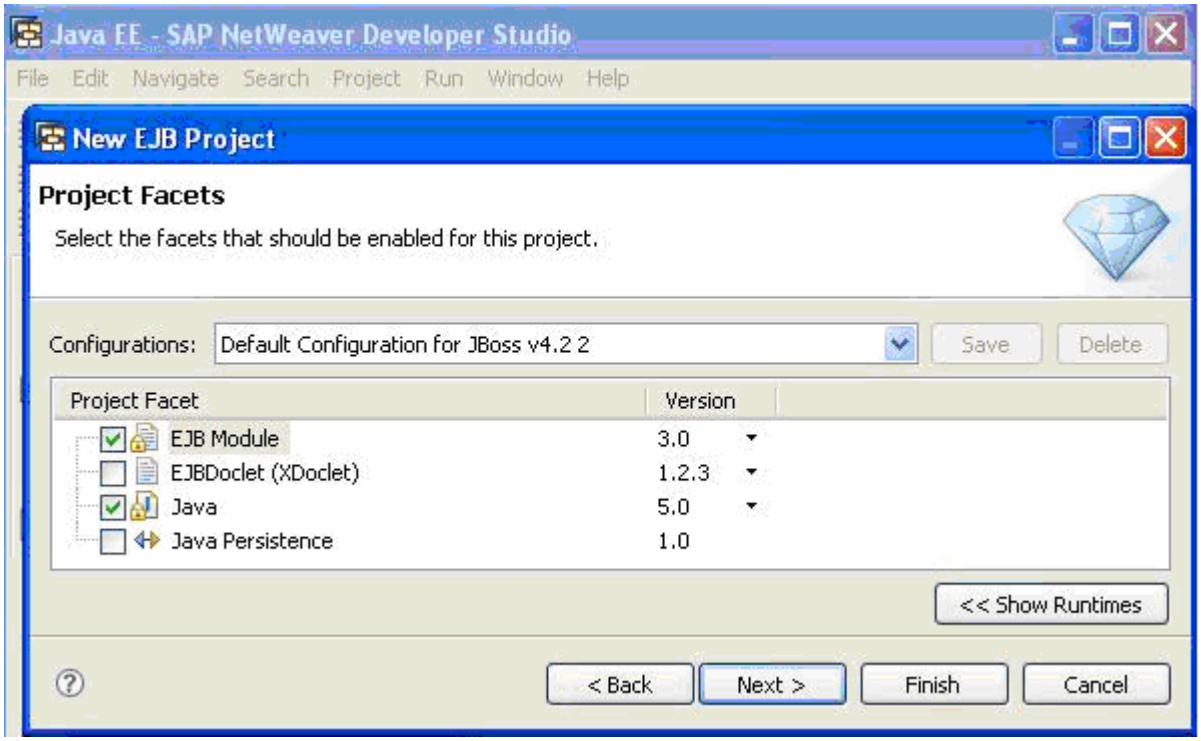
Choose jdk1.6.0_01 for JRE and the JBoss Home folder for the Application Server Directory. Click Finish.



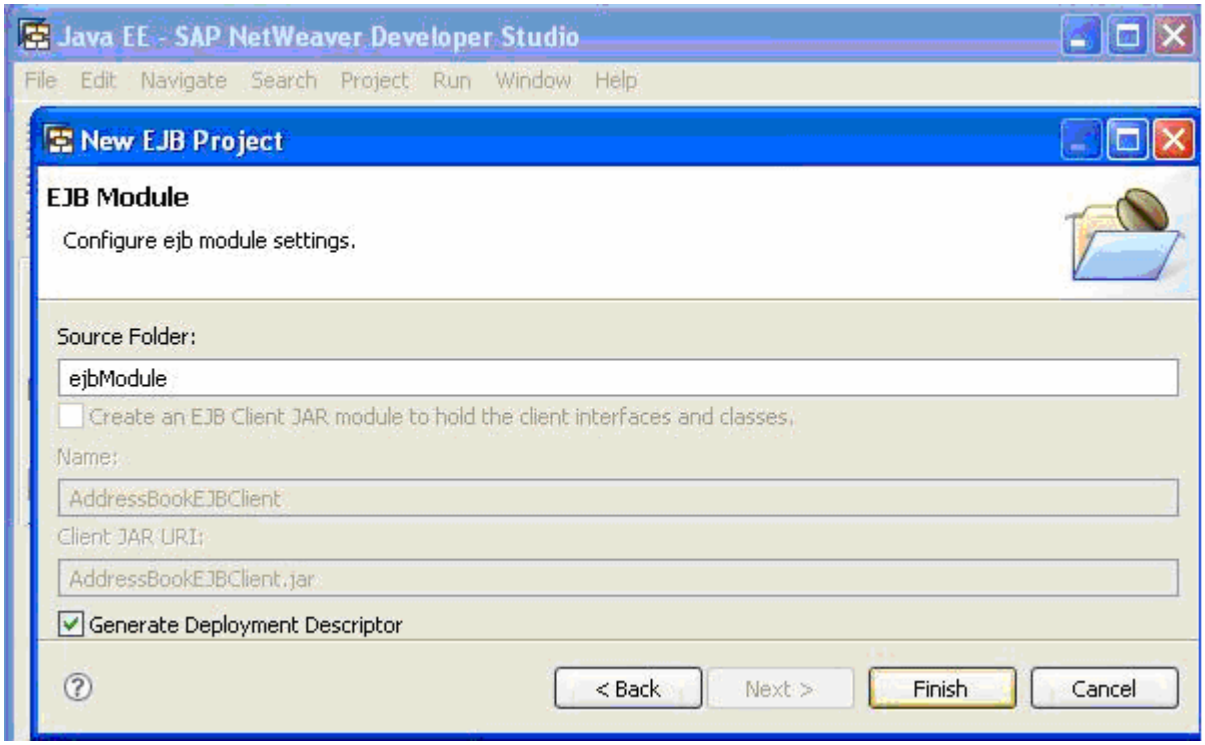
Select Default Configuration for JBoss v4.2.2.



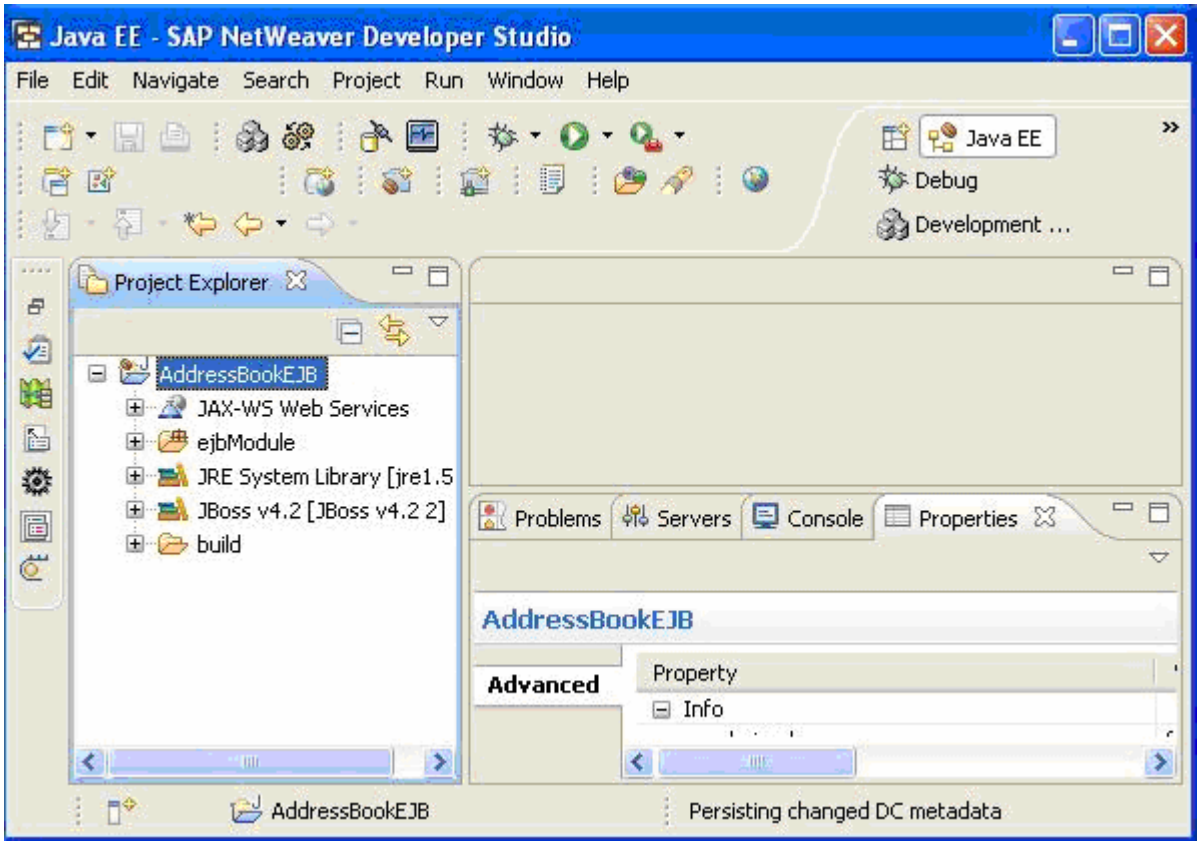
Select EJB Module version 3.0 and Java version 5.0 for Project Facets. Click Next.



Check Generate Deployment Descriptor option and click Finish.



The AddressBookEJB project node is displayed in the Project Explorer view.



EJB Code Sample

Class: ContactDTO

```
package com.hcl.ab.dto;

import java.io.Serializable;

public class ContactDTO implements Serializable {
		private static final long serialVersionUID = 1L;
		private String id;
		private String name;
		private String emailId;

		public ContactDTO(String id, String name, String emailId) {
		    this.id = id;
		    this.name = name;
		    this.emailId = emailId;
		}

		public String getId() { return id; }
		public void setId(String id) { this.id = id; }
		public String getName() { return name; }
		public void setName(String name) { this.name = name; }
		public String getEmailId() { return emailId; }
		public void setEmailId(String emailId) { this.emailId = emailId; }

}
```

Class: ContactCache

```
package com.hcl.ab.util;

import java.util.HashMap;
import java.util.Map;
import com.hcl.ab.dto.ContactDTO;

public class ContactCache {

		private static Map<String, ContactDTO> CONTACTS = new HashMap<String, ContactDTO>();

		public static ContactDTO findContact(String contactId) {
		    return CONTACTS.get(contactId);
		}
```

```
        public static Map<String, ContactDTO> findAllContacts() {
            if (CONTACTS.isEmpty()) {
                    ContactDTO contact = new ContactDTO("1", "Tom Sawyer", "tom.sawyer@test.in");
                    CONTACTS.put(contact.getId(), contact);
                    contact = new ContactDTO("2", "Huck Finn", "huck.finn@test.in");
                    CONTACTS.put(contact.getId(), contact);
            }
            return CONTACTS;
        }

        public static void addContact(ContactDTO contact) {
            CONTACTS.put(contact.getId(), contact);
        }

        public static void updateContact(ContactDTO contact) {
            CONTACTS.put(contact.getId(), contact);
        }

        public static ContactDTO removeContact(String contactId) {
            return CONTACTS.remove(contactId);
        }
}
```

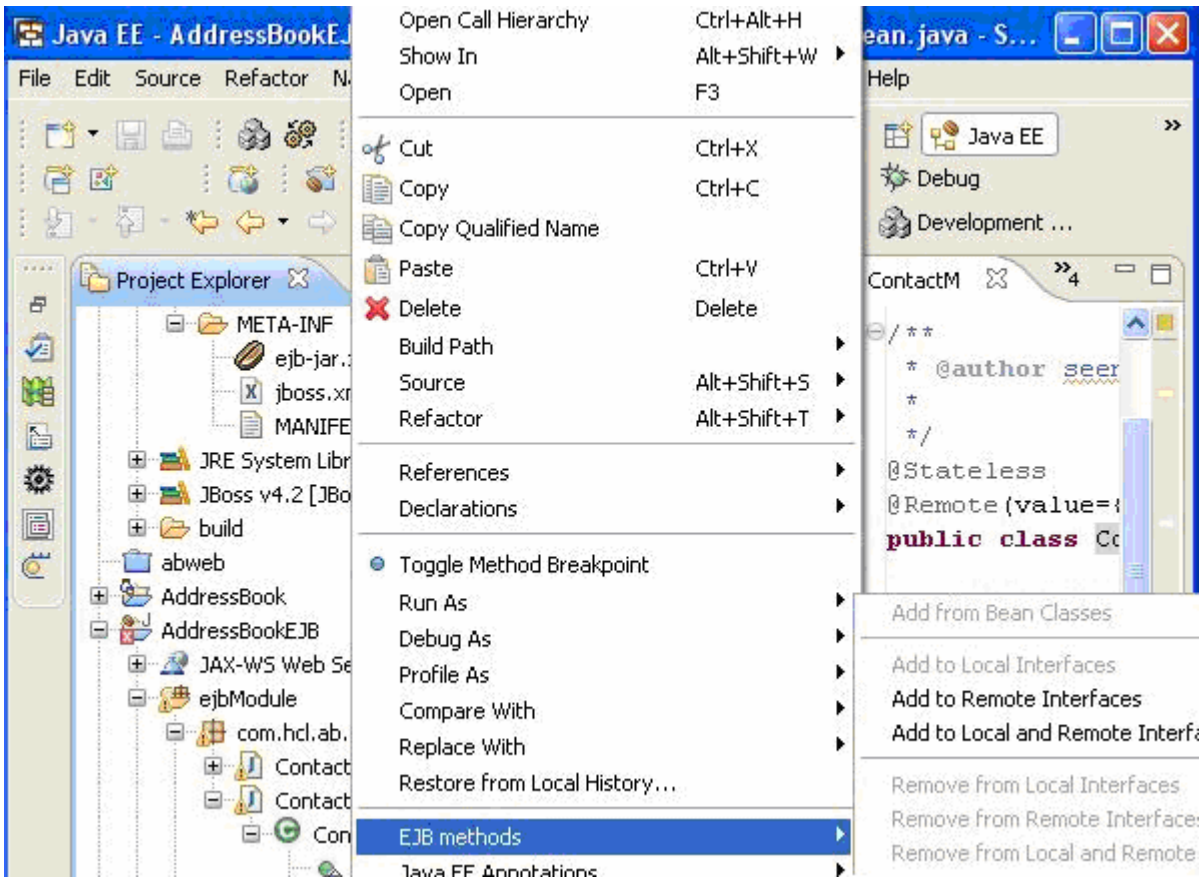
Class: ContactMgrBean

Create an empty interface named ContactMgr. Create a class named ContactMgrBean which implements ContactMgr. Include the annotations for the Stateless bean as shown in the code snippet below. NetWeaver Developer Studio Code Assist is available for annotations as well.

```
package com.hcl.ab.bo;

import java.util.Map;

import javax.ejb.Remote;
import javax.ejb.Stateless;

import com.hcl.ab.dto.ContactDTO;
import com.hcl.ab.util.ContactCache;

@Stateless
@Remote(value={ContactMgr.class})
public class ContactMgrBean implements ContactMgr {

        public void addContact(ContactDTO contact) {
```

```
            ContactCache.addContact(contact);
        }

        public void updateContact(ContactDTO contact) {
            ContactCache.updateContact(contact);
        }

        public ContactDTO removeContact(String contactId) {
            return ContactCache.removeContact(contactId);
        }

        public ContactDTO findContact(String contactId) {
            return ContactCache.findContact(contactId);
        }

        public Map<String, ContactDTO> findAllContacts() {
            return ContactCache.findAllContacts();
        }
}
```

Right click the methods in the ContactMgrBean. Choose EJB methods > Add to Remote Interfaces. The method gets added to the Remote interface. Repeat the same for all methods.



Deployment Descriptor

ejb-jar.xml

Include the following snippet in ejb-jar.xml

```
<enterprise-beans>
    <session>
        <ejb-name>ContactMgrBean</ejb-name>
        <mapped-name>ContactMgrBean</mapped-name>
        <business-remote>com.hcl.ab.bo.ContactMgr</business-remote>
        <ejb-class>com.hcl.ab.bo.ContactMgrBean</ejb-class>
        <session-type>Stateless</session-type>
        <transaction-type>Container</transaction-type>
    </session>
</enterprise-beans>
<assembly-descriptor>
    <container-transaction>
        <method>
            <ejb-name>ContactMgrBean</ejb-name>
            <method-name>*</method-name>
        </method>
        <trans-attribute>Supports</trans-attribute>
    </container-transaction>
</assembly-descriptor>
```

jboss.xml

Create a file called jboss.xml. Include the following snippet.

```
<jboss>
    <enterprise-beans>
        <session>
            <ejb-name>ContactMgrBean</ejb-name>
            <jndi-name>ContactMgrBean</jndi-name>
        </session>
    </enterprise-beans>
</jboss>
```

EJB Client Code Sample

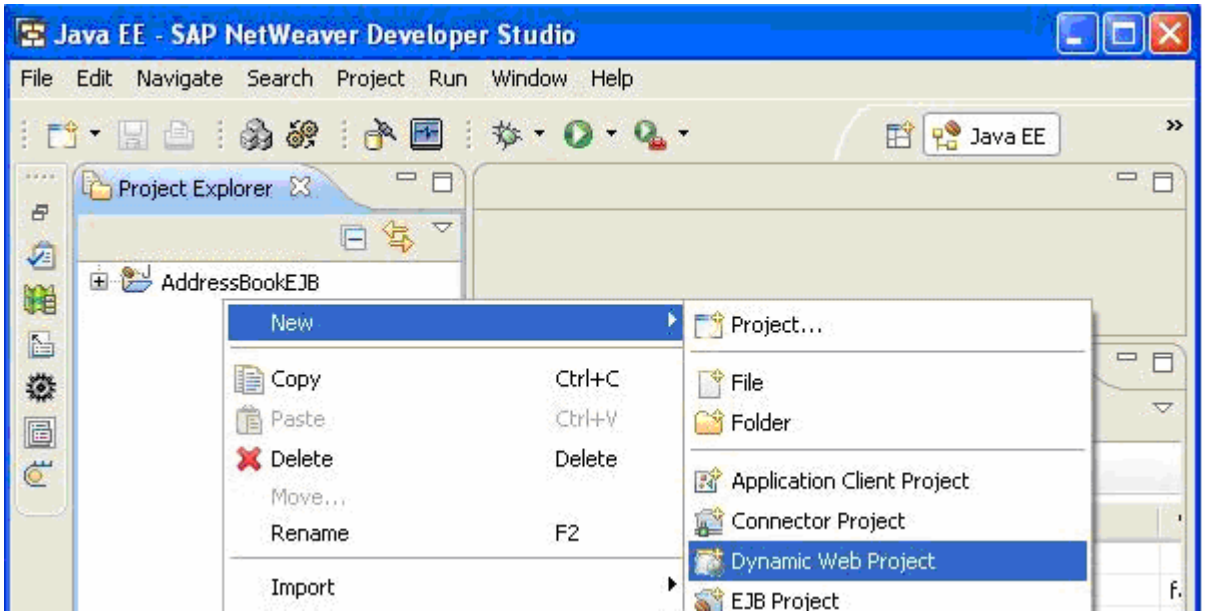
The following snippet contains the Client code that accesses the ContatcMgrBean. This sample can be run directly from the NWDS IDE after the application has been published

```
package com.hcl.ab.client;


import java.util.Properties;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import com.hcl.ab.bo.ContactMgr;
import com.hcl.ab.dto.ContactDTO;

public class ContactMgrClient {
                public static String CONTACT_BEAN = "ContactMgrBean";
                public static String INITIAL_CONTEXT_FACTORY = "org.jnp.interfaces.NamingContextFactory";
                public static String PROVIDER_URL = "jnp://localhost:1099";

                public static void main(String args[]) {
                    Properties properties = new Properties();
                    properties.setProperty(Context.INITIAL_CONTEXT_FACTORY, INITIAL_CONTEXT_FACTORY);
                    properties.setProperty(Context.PROVIDER_URL, PROVIDER_URL);
                    try {       Context ctx = (Context) new InitialContext(properties);
                                ContactDTO contact = new ContactDTO("3", "Arihant", "arihant@tirt.in");
                                ContactMgr contactMgr = (ContactMgr) ctx.lookup(CONTACT_BEAN);
                                contactMgr.addContact(contact);
                                System.out.println(contactMgr.findContact("1").getName());
                    } catch (NamingException e) { e.printStackTrace(); }
                }
}
```

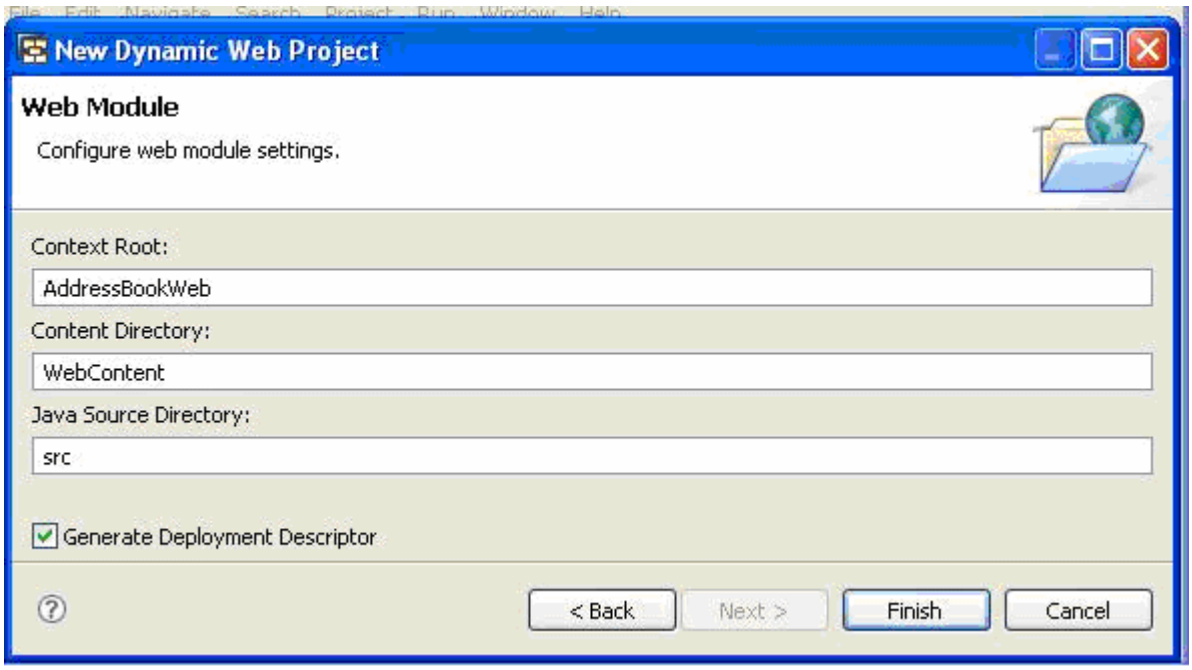
Developing Web Application

Creating a new Web Project

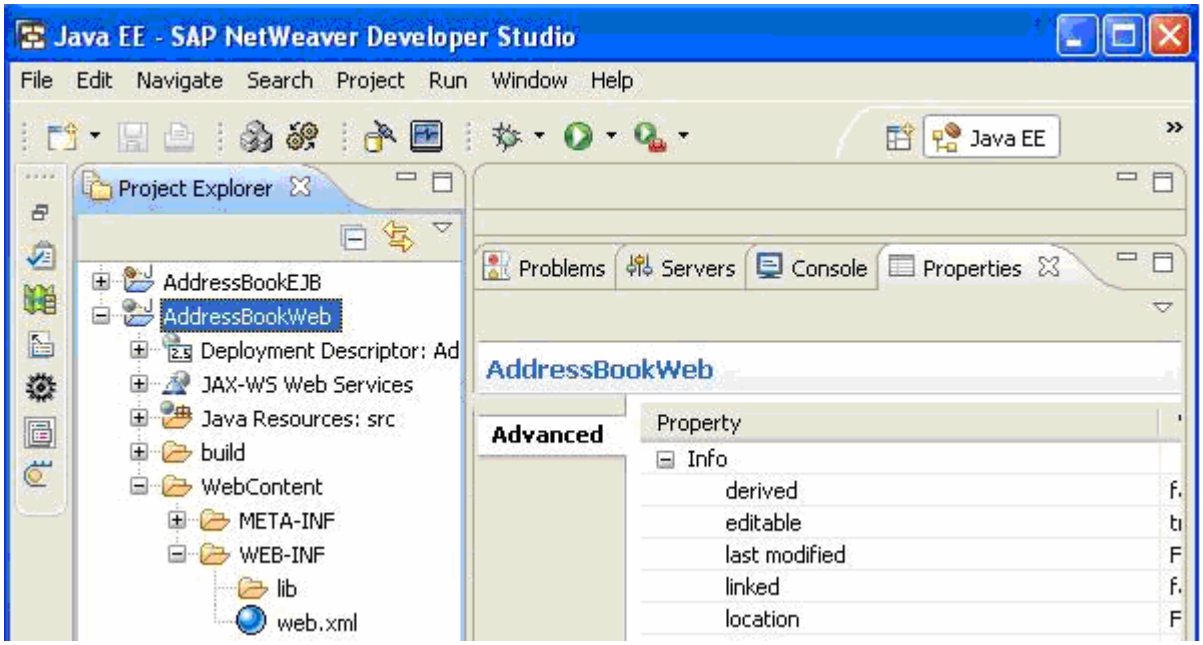
Right click on project explorer view and choose New > Dynamic Web Project.



Enter Context Root as AddressBookWeb. Check Generate Deployment Descriptor option. Click Finish.



The newly created AddressBookWeb project is displayed in the Project Explorer.



JSP Code Sample

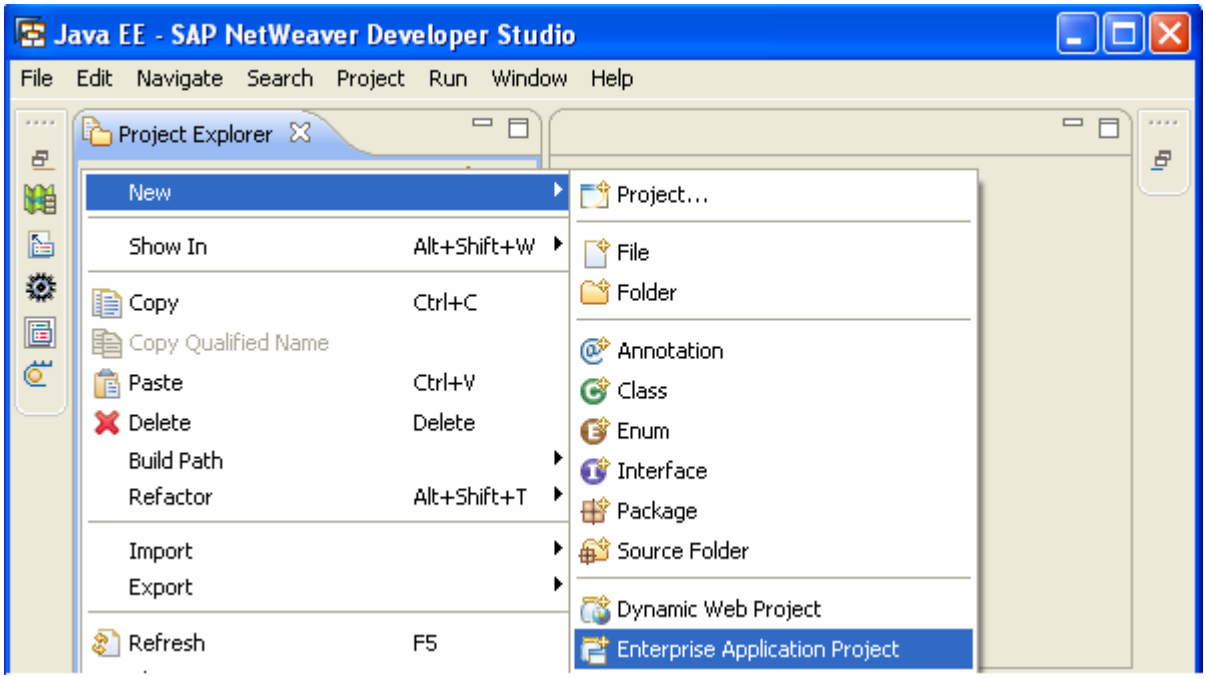
```
package com.hcl.ab.client;
<%@page language="java" contentType="text/html; charset=UTF-8"
                pageEncoding="UTF-8" import="javax.naming.InitialContext,          com.hcl.ab.bo.ContactMgr,
                com.hcl.ab.dto.ContactDTO,  java.util.Iterator,java.util.Collection"%>

<%!private ContactMgr contactMgr = null;
                private static String CONTACT_MGR_BEAN = "ContactMgrBean";

                public void jspInit() {
                    try {
                                InitialContext ctx = new InitialContext();
                                contactMgr = (ContactMgr) ctx.lookup(CONTACT_MGR_BEAN);
                    } catch (Exception e) {
                                e.printStackTrace();
                    }
                }%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<table border="1">
```

```
                <%
                    Collection contacts = contactMgr.findAllContacts().values();
                    for (Iterator contactIter = contacts.iterator(); contactIter.hasNext();) {
                            ContactDTO contact = (ContactDTO) contactIter.next();
                %>
                <tr>
                    <td><%=contact.getName()%></td>
                    <td><%=contact.getEmailId()%></td>
                </tr>

                <%} %>
</table>
</body>
</html>
```

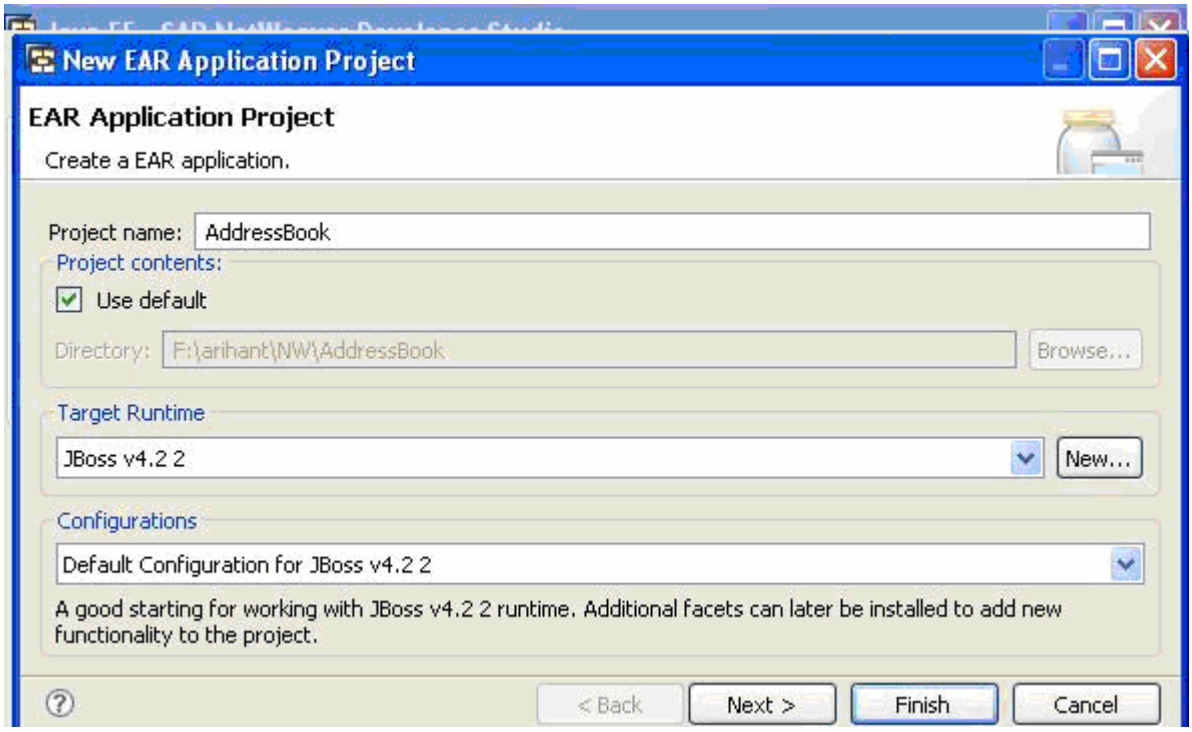
Developing Enterprise Application

Creating an EAR Project

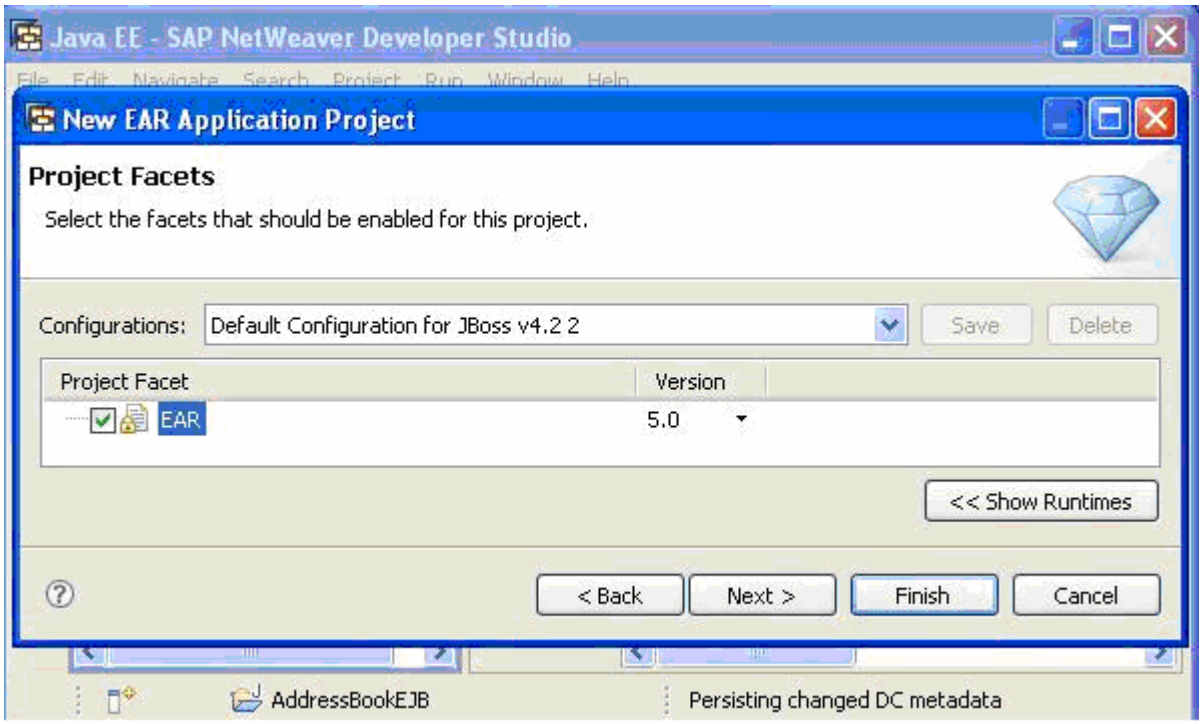
Right click on project explorer view. Choose New > enterprise Application Project.



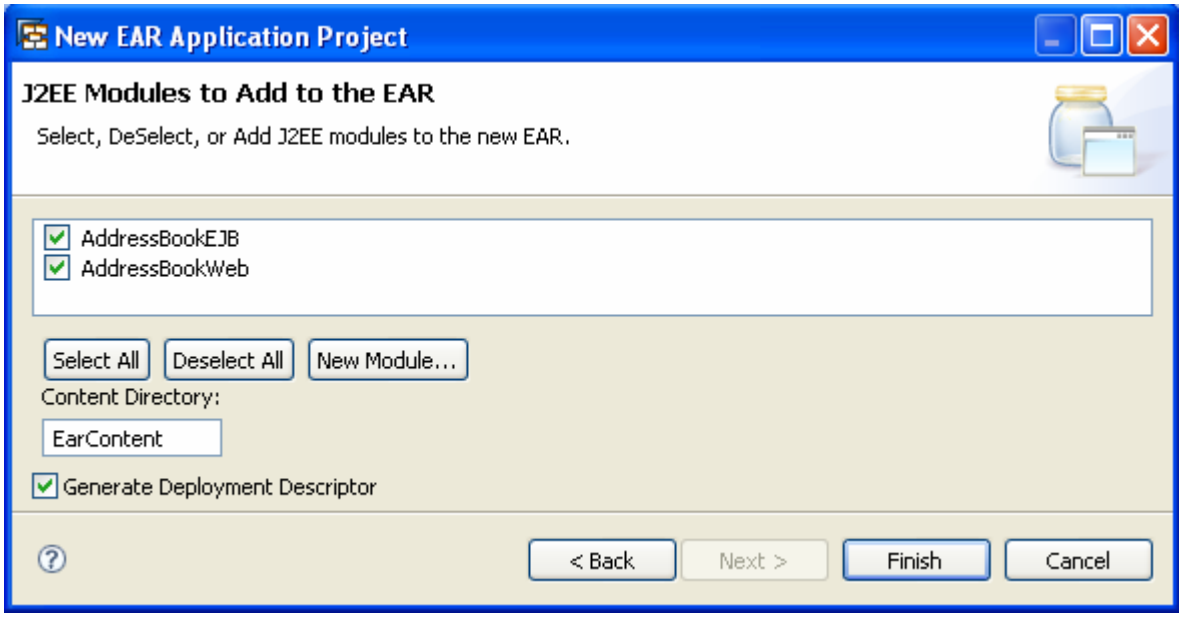
Enter the Project name as AddressBook. Choose JBoss v4.2 2 as the Target Runtime. Choose Default Configuration for JBoss v4.2 2 for Configurations. Click Next.



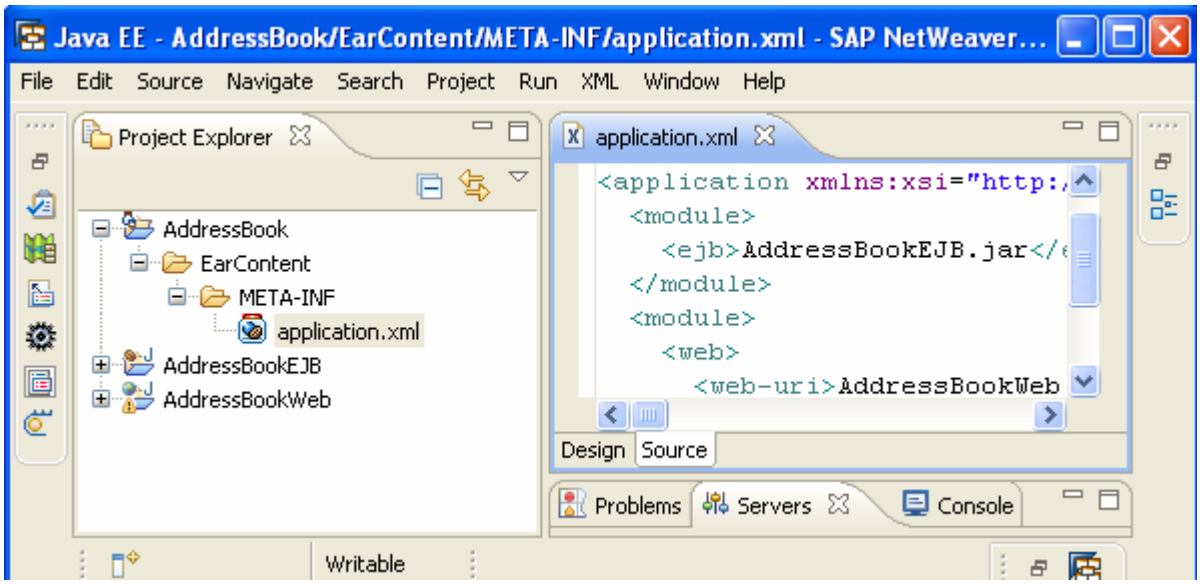
Select Project Facet as EAR 5.0. Click Next.



Add AddressBookEJB and AddressBookWeb modules to the EAR. Click Finish.

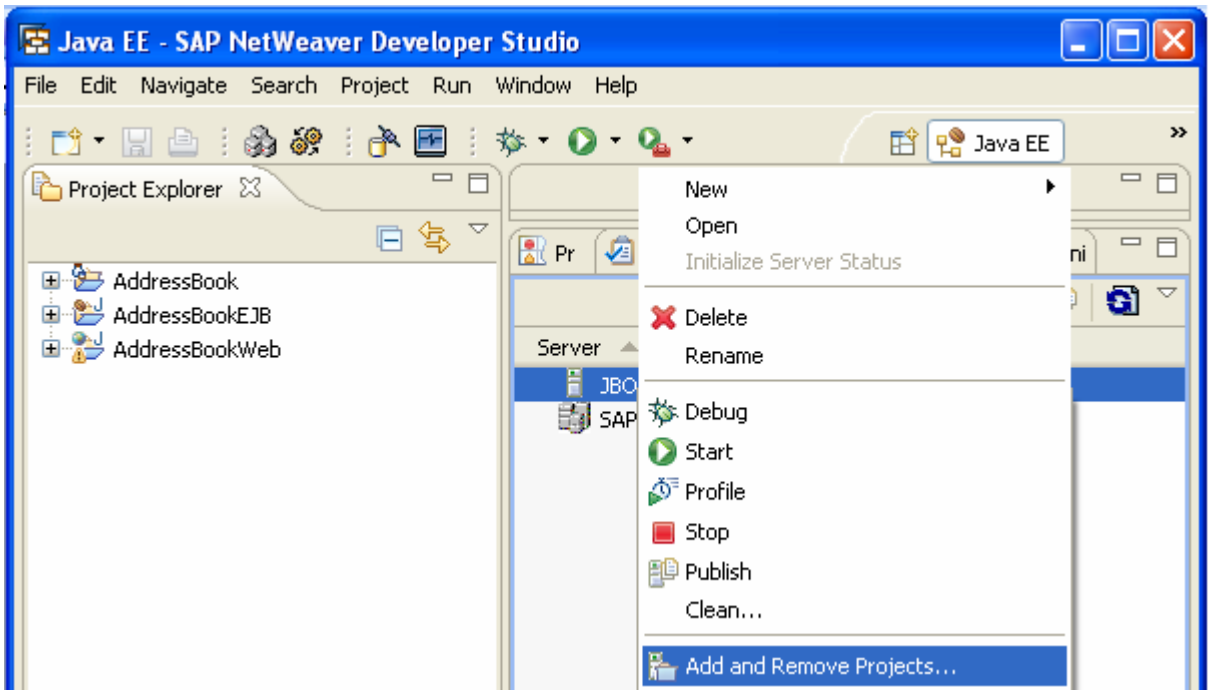


The AddressBook EAR project is created successfully. The EAR deployment descriptor application.xml contains ejb and web modules.

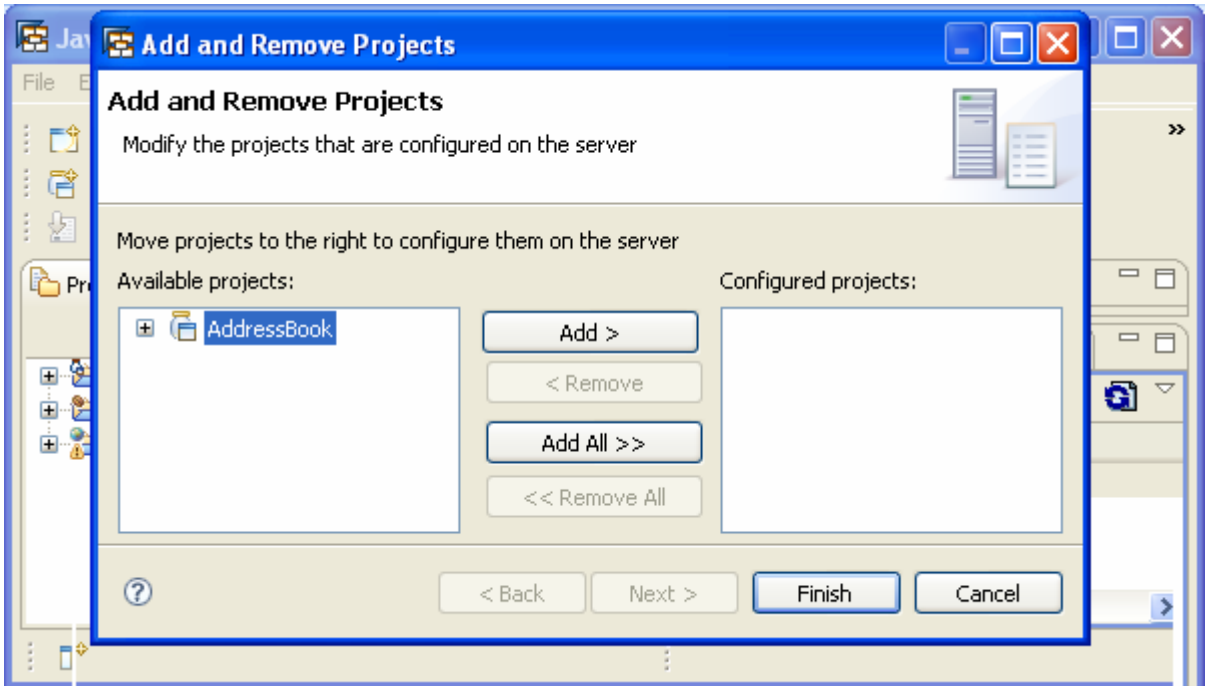


Publishing J2EE Application

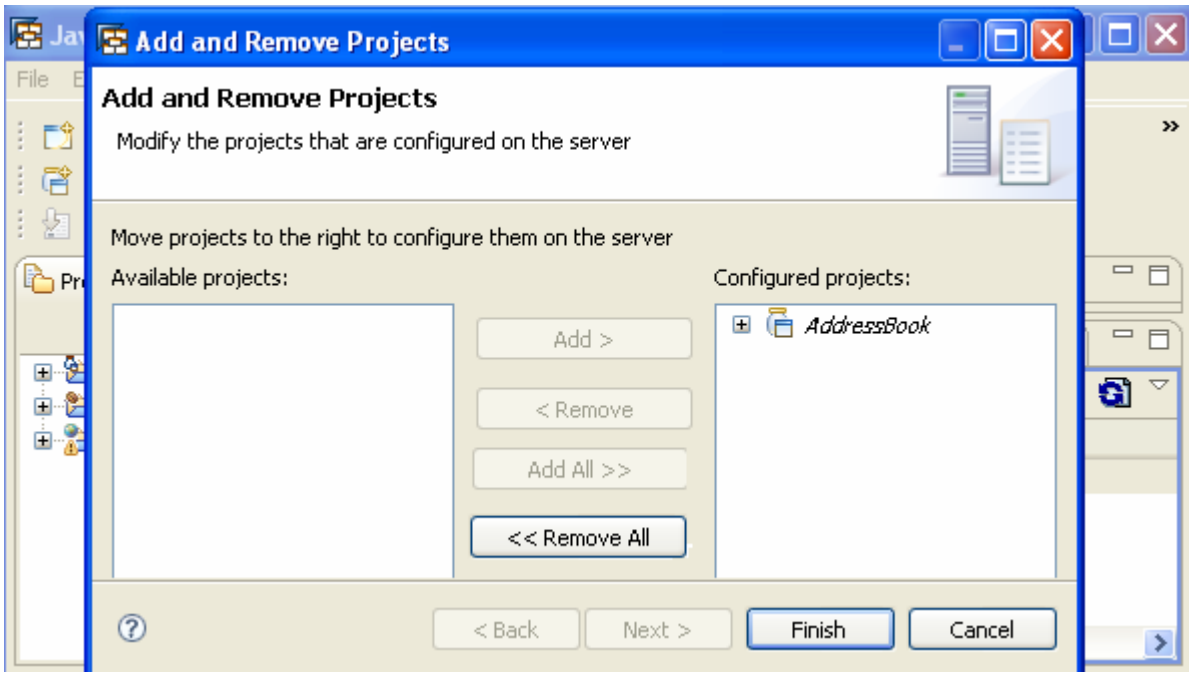
Right click the JBoss node in Server View. Choose Add and Remove Projects in the context menu.



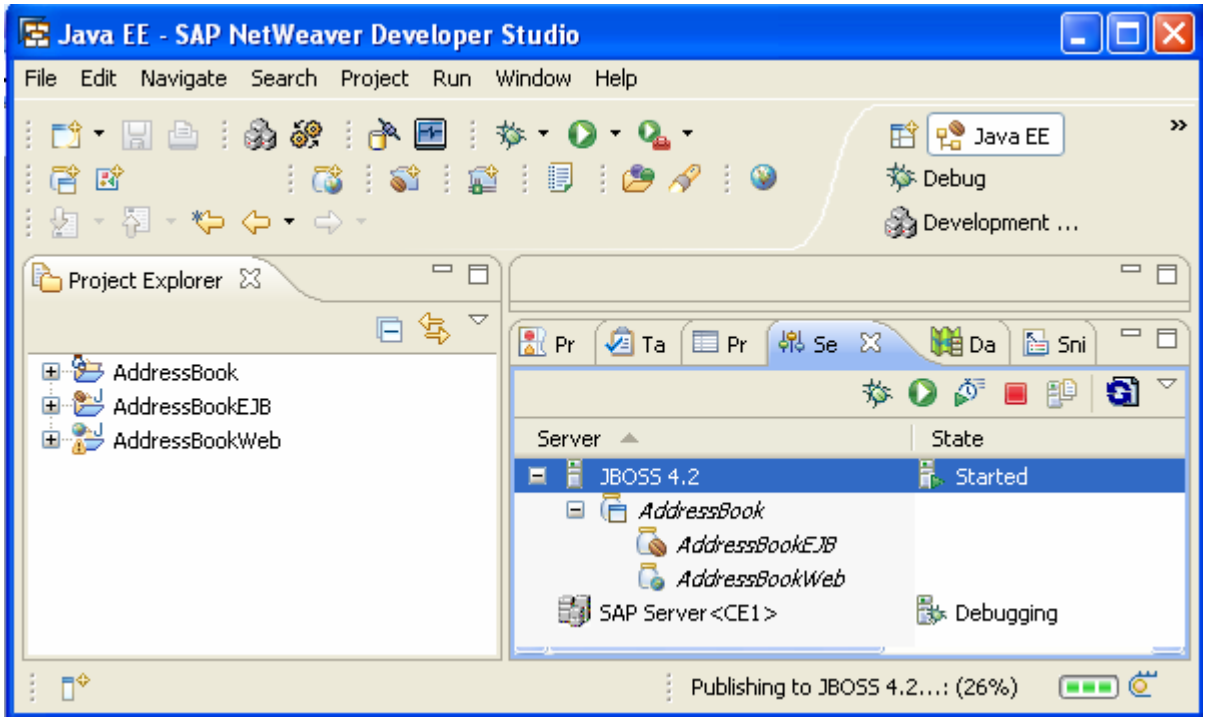
Select the AddressBook EAR project. Click Add.



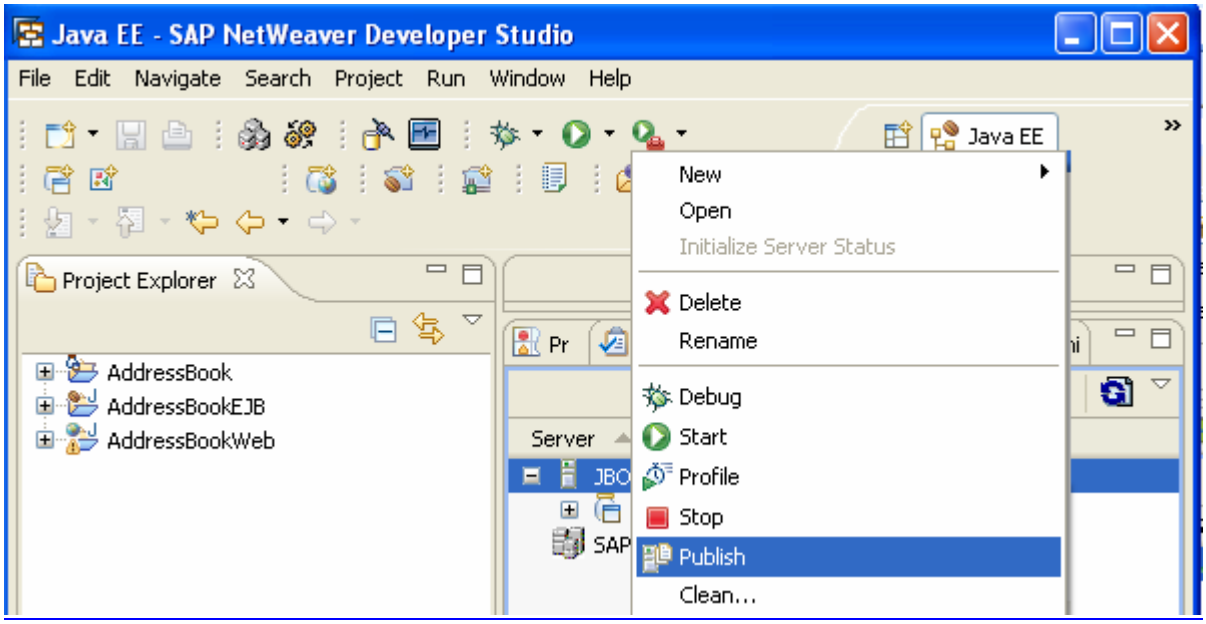
The AddressBook project is moved to the configured projects list.



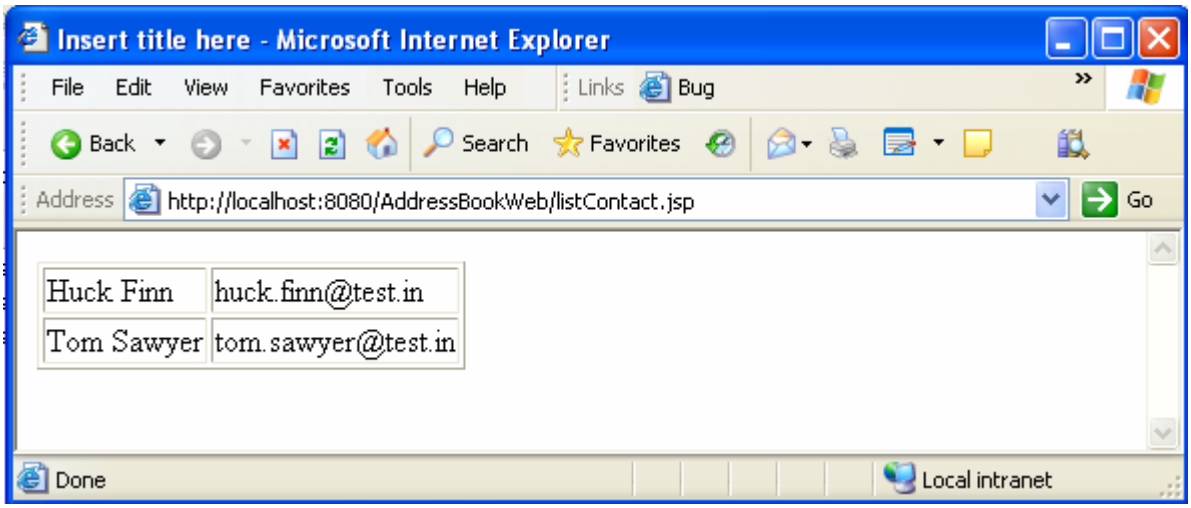
The AddressBook project is displayed under the JBoss node.



Right click on the JBoss 4.2 node. Click Publish. The application gets deployed in the server.



Access the URL http://localhost:8080/AddressBookWeb/listContact.jsp. The List contacts page is displayed with the contacts.



Copyright