

# *Novell eCommerce Beans*

Welcome to the Novell eCommerce Beans course. This course is intended to provide you with the information you need to create Web applications integrated with eDirectory using Novell's eCommerce JavaBeans.

## **Contents:**

- Course Purpose
- What is a Directory?
- Novell eCommerce Beans Overview
- Sessions
- Novell eCommerce Beans Introduction
- LDAP Beans
- Lab Setup
- Use HTML and a Servlet to Build an Authentication Page
- Add Other Methods of Authentication
- Allow the User to Change Attributes in the Directory
- Display the Attributes for the User to See
- Establish Secure Connection and Unauthenticate from eDirectory
- Security Beans
- Create an eCommerce Application (Optional Lab)
- Conclusion

## Course Description Table

<b>Course Description</b>	This course will help you integrate Web applications with eDirectory.
<b>Objective</b>	You will learn the fundamentals of Web programming and how to create objects in eDirectory using eCommerce Beans.
<b>Estimated time to complete this course</b>	It will take about 4 hours to complete this course.
<b>Prerequisites</b>	<ul style="list-style-type: none"> <li>• Basic HTML skills</li> <li>• Intermediate Java programming skills</li> <li>• Basic understanding of an LDAP Directory and Novell eDirectory</li> <li>• Basic eDirectory familiarity.</li> </ul>
<b>Required Items</b>	See "Required Setup"
<b>Optional Items</b>	None
<b>Required Setup</b>	<p>To complete the lab exercises:</p> <ul style="list-style-type: none"> <li>• A workstation with your favorite Java compiler</li> <li>• The JDK from Sun, available at <a href="http://java.sun.com/j2se/1.4/download.html">http://java.sun.com/j2se/1.4/download.html</a>. (J2SE 1.4.0 is used in this course, but v1.2 or later should be fine.)</li> <li>• eDirectory version 8.6.2, available at <a href="http://www.novell.com/download">http://www.novell.com/download</a>. (eDirectory 8.6.2 on NetWare 6.0 is used in this course, but eDirectory on another platform will be fine.)</li> <li>• Novell eCommerce Beans available at <a href="http://developer.novell.com/ndk/downloadaz.htm">http://developer.novell.com/ndk/downloadaz.htm</a>.</li> <li>• A network connection to your eDirectory server.</li> </ul>
<b>Development Environment</b>	The software in this course was tested using a Pentium II machine with a 400 Mhz processor running Windows 2000 Pro with SP2. The NetWare server is a Pentium III machine with a 500 Mhz processor running NetWare 6 SP1. I used the Apache Web server and Tomcat application server. Certificate Server was the only additional service installed.

# Course Purpose

The purpose of this course is to teach you how to use Novell eCommerce Beans to integrate Web applications with eDirectory. This course covers:

- What is a Directory?
- Novell eCommerce Beans Overview
- Sessions
- Novell eCommerce Beans Introduction
- LDAP Beans
- Lab Setup
- Use HTML and a Servlet to Build an Authentication Page
- Add Other Methods of Authentication
- Add a Link to Create a New User
- Allow the User to Change Attributes in the Directory
- Display the Attributes for the User to See
- Establish Secure Connection and Unauthenticate from eDirectory
- Security Beans
- Create an eCommerce Application (Optional Lab)
- Conclusion

In order to fully understand the capabilities of Novell eCommerce Beans, it is important to understand several fundamental concepts first. In the first few sections we introduce basic concepts of an LDAP directory, eDirectory; J2EE fundamentals like Servlets, JSP, MVC architecture, and Beans; and Web programming fundamentals. If you have a good understanding of these topics, skip to the Novell eCommerce Beans Introduction section.

# *What is a Directory?*

A Directory is a database of objects on the network. Each object represents a physical object in the real world. Examples of objects that exist in a Directory are:

- user objects
- user group objects
- printer objects
- server objects

Each object in the Directory has properties or attributes and values associated with these properties. Examples of properties for a user object include:

- user's name, first and last
- email address
- mailing address
- department
- manager
- phone number
- rights or privileges the user has to other objects in the Directory

Some attributes are required for certain objects and others do not need a value. If the employee Joe Smith works for the Operations department of Novell, his user object in the Directory could be JSmith, with a property name department and property value of Operations.

Each Directory object has a different set of properties associated with its object. The Directory Schema governs the types of objects, object properties, and property values allowed in the Directory. You can see a list of objects for the Directory by looking at the Administration tool provided with the Directory or the Schema documentation provided with the Directory.

Examples of Directories include Novell's eDirectory, Microsoft's Active Directory, Sun's ONE Directory, and others.

# LDAP

Lightweight Directory Access Protocol (LDAP) is an open standard and the common protocol used to access Directories. In order to allow software developers to write applications to access the Directory via the LDAP protocol Directories need to support the LDAP protocol. You can request or submit data to the Directory through LDAP. For more information about LDAP, see <http://www.ldapzone.com>.

## Directory Objects

Two classifications of objects exist in an LDAP compliant Directory:

- Leaf objects
- Container objects

Leaf objects are objects such as user, user group, workstation, and printer objects.

Container objects are objects that contain other objects and logically organize other objects in the Directory. Container objects include Country objects, Organization objects, and Organizational Unit objects. The Schema defines which objects a container object can contain.

## Root Object

The Root object is a special kind of directory object that defines the top of the tree. All other container and leaf objects reside underneath the Root object.

## Naming Convention

The LDAP protocol defines a naming convention for objects in an LDAP Directory tree. Be aware that the naming convention syntax in LDAP differs from eDirectory. We will use the LDAP naming convention in our programs.

The LDAP naming convention uses the common name of the object together with the context. The context of an object is the position of the object in the Directory tree. It allows the Directory to find the object. The common name of the object is the name of the object in the Directory. In Figure 1, user jDesman is the common name for that object. His distinguished LDAP name that you would use (if you were making an LDAP request for his object) is: cn=jDesman, ou=dev, ou=prv, o=novell. This name comes from the name attributes of his objects. The two container objects, Dev and Prv, are Organizational Unit objects. Novell is the Organization object.

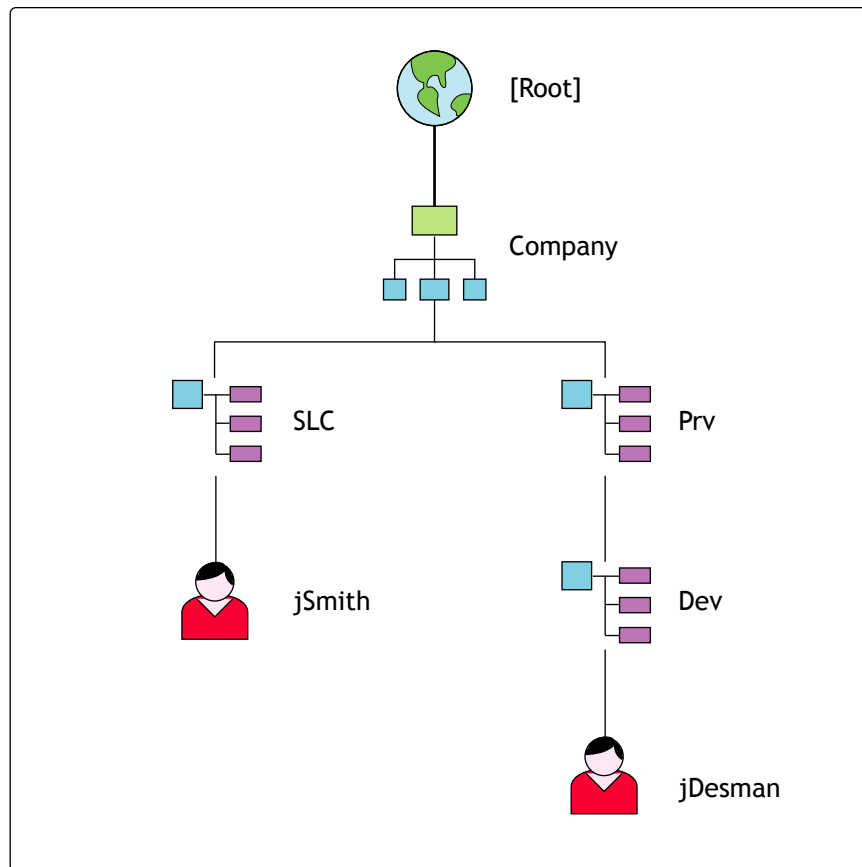


Figure 1: LDAP naming conventions.

For more information on LDAP naming, see <http://www.ldapzone.com>.

## eDirectory

eDirectory is a full service LDAP version 3 compliant Directory. It manages every resource on the network instead of each server managing the resources connected to it.

### eDirectory: From a Software Developer's Perspective

The eDirectory schema is extensible, meaning that you can add new classes or attributes to the schema depending on your application needs. eDirectory provides multiple interfaces from multiple languages and platforms. LDAP is the preferred connection protocol to eDirectory so the eDirectory interfaces support LDAP. As a developer, you have access to eDirectory from VB, C, C++, Java, PHP, Perl, or any interface that has LDAP classes or support. Novell has developed LDAP libraries for C, and Java. ActiveX controls have been developed for VB. LDAP classes for Java have been developed, as well as LDAP extensions for JNDI, eCommerce JavaBeans, and an LDAP JDBC driver.

As an eDirectory developer, you should be familiar with the schema of the Directory. Each API does have ways for you to programmatically see the schema of the Directory. An easier solution is to download the eDirectory Schema Reference from <http://developer.novell.com>. This document, provides information on attributes for each type of object in the Directory and is a significant resource for the Directory schema.

## Self-Check

1. Which of the following languages can be used to program to eDirectory?
  - a. Java
  - b. C/C++
  - c. Visual Basic
  - d. Perl
  - e. PHP
  - f. All of the above
2. Which object is the top of an eDirectory?  
\_\_\_\_\_
3. Name the protocol that is commonly used to access compliant Directories.  
\_\_\_\_\_
4. What is the LDAP distinguished name for Jsmith's user object?
  - a. Jsmith
  - b. Cn=jsmith. ou=slc. o=company
  - c. .cn=jsmith. ou=slc. o=company
  - d. ,cn=jsmith, ou=slc, o=company
  - e. cn=jsmith, ou=slc, o=company

---

**Answers:**

- 1) f. All of the above
- 2) Root or [Root]
- 3) LDAP, Lightweight Directory Access, or Lightweight Directory Access Protocol
- 4) e. cn=jsmith, ou=slc, o=company

# *Novell eCommerce Beans Overview*

Novell eCommerce Beans have been developed to facilitate Web application development. eCommerce Beans support any LDAP v2 or v3 compliant Directory so you can integrate your Web applications with any LDAP Directory. eCommerce Beans have been written to encapsulate a single task in one bean. The beans are simple to learn and implement. Here are some features you can add to your application using Novell eCommerce Beans:

- User Authentication
- User Profiling
- User Creation
- User Portal
- User Rights
- Secure Connections

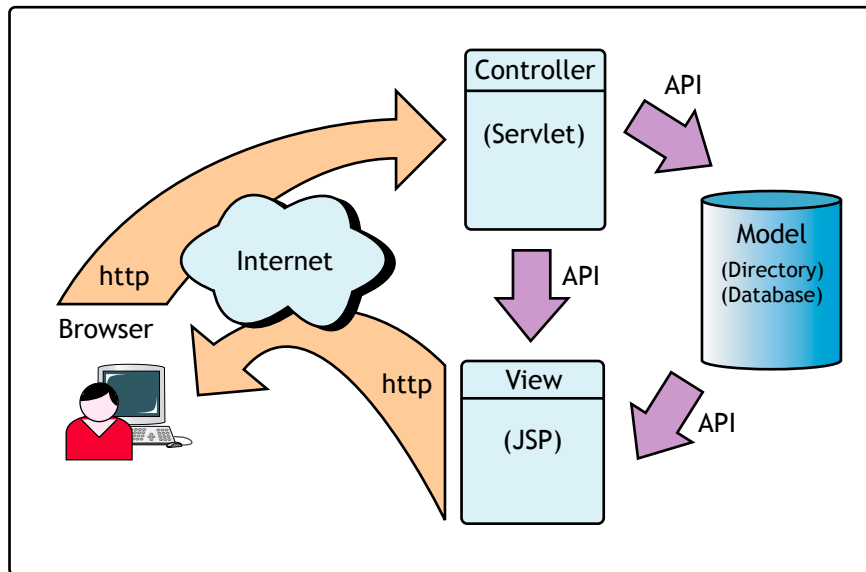
All of the information can be stored in the Directory. With the ability to leverage the existing attributes already defined for objects in eDirectory and the ability to create additional attributes, eDirectory makes using Novell eCommerce Beans powerful, flexible, and easy to use. That is the real power of eCommerce Beans.

## **Web Programming**

Web programming includes a variety of technologies from many vendors. In this course, I'll focus on how to integrate Web programming with Java Web technologies and architectures.

### **Architecture**

Web Programming with Java offers a little different architecture than desktop application programming. In addition to a browser and HTTP requests, we have a Java Application server that works together with the Web server to service requests for Web applications. The following diagram illustrates a commonly used Web programming architecture.



**Figure 2:** Model View Controller (MVC) architecture.

This architecture is referred to as the Model View Controller (MVC) architecture. The MVC architecture is a logical representation of how to implement a Java Web application. It is a conceptual architecture. Each component represents a function in the architecture. In Figure 2 I have labeled the common Web implementations of each component.

*Model*— The model is the datastore of the application. It is where the information the user is seeking is stored. The other components are the interface for the user to request and receive the information that is stored in the model. In a Web environment, we typically use a database or a Directory. The implementation of the model in our example is eDirectory.

*Controller*— The controller component is the “engine” of the architecture. It queries or updates the model for information and controls the flow of information back to the user. It will most often provide the user interface as well. The implementation of the controller in our example is a Java Servlet.

*View*— The view component displays information from the model to the user. The model must communicate with the view so that information is always current. The implementation of the view in our example is HTML. It could also be straight HTML, JSP, XML with XSL, or another display technology.

## Technologies

In order to completely understand the power and flexibility of the Model View Controller architecture, you should be familiar with the J2EE technologies Java Server Pages (JSP) and Java Servlets.

*Java Servlets*— For convenience I will only talk about the `HttpServlet`. Java Servlets are Java's answer to CGI. Java Servlets are a Java class that extends the Servlet API. Java Servlets are used to process information. They can easily receive information from a Web page or applet and then process that information by querying or updating the datastore for the application. The servlet can maintain connections to a database or Directory. A servlet gives a programmer the advantages of a fully functional Java API and also the ability to output content to a static Web page using HTML or other Web display technology. Simply put, a servlet is Java code with HTML embedded within it.

Some advantages of using Java Servlets are:

*Portability*— Servlets have the portability of Java. With little or no modifications, a servlet can be compiled and deployed for multiple servers.

*Powerful*— Unlike applets, servlets do not have the same security restrictions because the processing happens on the server. Servlets are not executed on the client —so you have a fully developed Java API to leverage.

*Performance*— Servlets are multi-threaded by default which offers better performance than CGI since CGI scripts open a new process for each connection.

A servlet is an excellent solution for data processing. It can receive data, process it, and request data from other sources easily and provide excellent performance as well. It is more work to provide a complex display in a servlet. A servlet is great for information processing, but it is more work if you want to display the data in a fancy way. If you have data processing to do as well as render a complex HTML page, you may want to consider using JSP.

In order to run a servlet, you will need a Java Application server that will process the servlet. An application server works with the Web server to process the requests for Java Web applications. Examples of Java Application servers are Apache Tomcat, BEA WebLogic, IBM WebSphere, and SilverStream Application Server.

To deploy a servlet, you should copy the `.class` file to the specified directory on the Java Application server and then restart the server. You will then be ready to access your servlet.

Frequently, your servlet will only be a part of a larger J2EE archive file such as a CAR/WAR/EAR file. In this case, copy the archive file to the specified directory and restart the application server.

To access your servlet, type in the fully qualified URL to the server including port number and the path to the servlet. If you choose the default deployment directory, the URL will most likely have the servlet path before it. The default port is 8080, but can be changed to a different port. For example, in order to access a servlet called `Servlet1`, on the server `AppServer.Novell.com`, type the following URL: `http://www.AppServer.Novell.com:8080/servlet/Servlet1`.

Below is sample code from a basic “Hello World” servlet.

---

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class helloservlet extends HttpServlet
{
    private static final String CONTENT_TYPE = "text/html";
    public void init() throws ServletException
    {
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Hello World Servlet</title></head>");
        out.println("<body bgcolor=\"#ffc800\">");
        out.println("<h1>");
        out.println(request.getParameter("name") + " says Hello World!!</h1>");
        out.println("</body></html>");
    }
    public void destroy()
    {
    }
}
```

---

This servlet relies on the HTML page with a text box called “name” that submits the contents of the text box to the servlet. In order to do this, the <form> tag attributes need to be:

---

```
<form action="HTTP://host_name:8080/servlet/helloservlet" method="post">
```

---

## JavaServer Pages (JSP)

JavaServer Pages are an extension of Java servlets. Essentially, JSP is the reverse of a Java servlet. JSP is an HTML page with embedded Java code inside to provide the dynamic functionality to the Web page.

JavaServer Pages also require an application server in order to run. The application server actually compiles the JSP into a Java Servlet.

A JavaServer Page is deployed by copying the .jsp file to the application server directory just as with servlets. When the page is invoked from a browser, the JSP will display itself. Take a look at this sample JSP and notice how the JSP is the reverse of a Java servlet. You can use native HTML tags and the Java code is marked by the <% and %> signs. The advantage of JSP over servlets is that you have an easier HTML design capability instead of having to write an out.println() statement for every tag you need.

---

```
<HTML>
  <HEAD>
    <TITLE>Hello World JSP!!</TITLE>
  </HEAD>
  <BODY>
    <% out.println(response.getParameter("name"));%> says Hello World!
  </BODY>
</HTML>
```

---

Just like the Java servlet, this JSP relies on an HTML page with a text box called “name” that will post the information to this JSP.

JSP works well when you have a more intensive display to render than a Java servlet. JSP gives you the advantage of information processing together with the easy ability to display the information in an attractive way to the user. For example, if you have a complex HTML page to display, you would probably want to use JSP because it is easier to code and test the HTML in JSP.

JSP and JavaBeans can be used together to easily provide a powerful dynamic Web application.

## Self-Check

1. What is the Model component of MVC?
  - a. DataStore of the application, typically a database or Directory
  - b. The “engine” of the application
  - c. Presentation to the user
2. What is the View component of MVC?
  - a. DataStore of the application, typically a database or Directory
  - b. The “engine” of the application
  - c. Presentation to the user
3. What is the Controller component of MVC?
  - a. DataStore of the application, typically a database or Directory
  - b. The “engine” of the application
  - c. Presentation to the user

4. You would use a Java Servlet to process information and query a datastore.
  - a. True
  - b. False
5. Java Servlets take more time to execute than CGI.
  - a. True
  - b. False
6. Which technology can you type in raw HTML tags?
  - a. JSP
  - b. Servlet
7. Which of the following are ways to store the state of a Web application?
  - a. Cookies
  - b. URL Re-writing
  - c. Override the setState() method of a servlet to store the HTTP state
  - d. Sessions

---

**Answers:**

- 1) a. DataStore of the application, typically a database or Directory
- 2) c. Presentation to the user
- 3) b. The "engine" of the application
- 4) a. True
- 5) b. False
- 6) a. JSP
- 7) a. Cookies  
b. URL Re-writing  
d. Sessions

# Sessions

Sessions are important in eCommerce Web programming because they can be tracked. Let's begin with HTTP which is a stateless protocol. You don't have any way to store information that you need as the programmer in the protocol itself. This is very pertinent in, for example, an online store application. The Web server, via HTTP, cannot keep track of the user's online shopping cart. So how do you store information in a Web application?

A session is a way of identifying the user even through multiple page requests; something that HTTP doesn't have a mechanism to do. A session resides on the server so it handles the actual creation of the session for you assigning a unique ID to each session. Your job is to associate that session with the client who is making the HTTP request. This is normally implemented in one of two ways.

*Cookies*—The first way, and probably the most common, is to use a Cookie. In the Cookie, you can store information about the session that can be used later to identify the user. But be careful about what information you store in the Cookie. You should never store sensitive information like username, passwords, or credit card numbers.

This is an excellent solution because you have a Cookie interface in the package `javax.servlet.http.cookie` to use for your advantage. Normally, you would store the session ID inside the Cookie, and you can use the interface to request the Cookie and its attributes that are of interest to you.

*URL Rewriting*—Use URL rewriting to append information about the session to the URL allowing the server to identify which session belongs to the user. This is also an excellent way of implementing session tracking because this still allows you to track the user's session even if the user has disabled Cookies.

## Session Tracking API

Java has provided an easy to use Session tracking API. It basically involves five tasks.

1. Lookup the Session ID from the Cookie or URL associated with a specific user.
2. Create a new Session if necessary.
3. Lookup information associated with a Session.
4. Storing information about a Session.
5. Discarding a Session when it is completed.

Here are some common methods of the Session Tracking API. The methods are straightforward.

`getID()` - returns the Session ID

`isNew()` - returns true if the Session is new to the browser

`getCreationTime()` - returns the time the particular Session was created

`getLastAccessedTime()` - returns the time the Session was last accessed

`setAttribute()` - associates a value with a Session

`getAttribute()` - returns a value of an associated attribute

For more information and exact method signatures, see <http://java.sun.com>.

# Novell eCommerce Beans

## Introduction

Novell eCommerce Beans are used to develop Web-based Directory applications. They are 100% pure Java, with no dependency on native code. Because of this, they can be used in any J2EE technology such as JSP, Servlets, or Enterprise JavaBeans.

eCommerce Beans provide authentication to and read and write access to any LDAP Directory. You can perform tasks as creating objects, adding or modifying object attributes, setting or modifying object rights within the Directory. All this can be done through SSL. This gives you a lot of power as you develop your application. Since these beans are written entirely in Java, support LDAP, and provide security over SSL/TLS, they are a nice fit for Web applications.

All eCommerce Beans are included in three library files. As with normal libraries, they should be in your classpath and be imported into your application. The three library files are `ecb.jar`, `ecbldap.jar`, and `ecbsecurity.jar`. Depending on which classes you want to import into your application, the import statements should be statements such as:

```
import com.novell.ecb.*
import com.novell.ecb.ldap.*
import com.novell.ecb.security.*
```

eCommerce Beans provide a specific bean that will accomplish a simple task. In nearly all cases, only a single business logic task has been included in the bean. This is referred to as a Command Bean. A Command Bean implements the Command interface providing isolated components to your Web application. The Command interface includes several methods that you should be familiar with.

eCommerce Beans receive 3 methods from the Command interface.

eCommerce Beans Receive Methods	Explanation
<code>isReady()</code>	Tests to see if the required properties have been input and the bean is ready to execute. It returns a boolean value.
<code>execute()</code>	Executes the bean with the input properties that have been set.
<code>reset()</code>	Resets the beans properties. Must be called before the bean can be re-used.

eCommerce Beans also receive 3 states from the Command interface.

eCommerce Beans Receive States	Explanation
New	This is the state of the bean after instantiation. The isReady() method returns false the set methods should be called to set the input properties.
Initialized	The bean is initialized when the required input properties for execution have been set. The isReady() method returns true.
Executed	This is the state of the bean after execution. The get methods can be called to return the values needed. The reset method should be called before the bean is re-used.

eCommerce Beans work similarly to other JavaBeans you may have worked with. Each bean has its own set of properties. You must initialize the required properties in order for the bean to execute. Additional properties can be initialized to add functionality.

## Necessary Steps to Use eCommerce Beans

1. Instantiate the bean.

```
AuthenticateLdap bean = new AuthenticateLdap();
```

2. Set the required and any optional input properties.

```
bean.setURL("ldap://myserver.novell.com");
```

3. Call the execute method after the properties are set.

```
bean.execute();
```

4. Read any desired output properties of the bean.

```
boolean auth = bean.isAuthenticated();
```

Each bean is used following this same pattern. The documentation for the bean should tell you which properties are required, then the bean can be run. Or you can use the isReady() method to test yourself.

## Types of eCommerce Beans

Novell eCommerce Beans consist of two types of beans:

- LDAP Beans
- Security Beans

# LDAP Beans

LDAP beans provide read and write access to any LDAP Directory as well as authentication and rights assignments. All of this can be done over an encrypted channel using SSL. The following table lists several commonly used LDAP command beans and their functions.

LDAP Command Bean	Function
AuthenticateLdap	To Authenticate to an LDAP Directory.
LdapConnection	To provide a communication channel between Web application and the Directory. Use anytime you intend to do more in the Directory besides Authenticate.
ContextlessLoginLdap	To login specifying the user's fully distinguished name. You can also set a filter to search and login by a user's specific attribute such email address.
CreateLdapEntry	Create an entry in an LDAP Directory.
DeleteLdapEntry	Delete an entry from the Directory.
ListLdapEntry	Lists the entries in the Directory at a specified context.
ModifyLdapEntry	Used to modify the attributes of a entry in the Directory. You can add a new attribute to the object, replace an attribute, or delete one as well as modify the rights of an entry.
ChangePasswordLdapEntry	To change the password of an entry in the Directory.
SetPasswordLdapEntry	Sets the password for an entry.
ReadLdapEntry	To read the attributes of an LDAP entry.
SearchLdapEntry	Searches through the Directory to find the entry with a specified property value.
UnauthenticateLdap	Unauthenticates from the Directory.

## Develop with LDAP Beans

The following six steps will be the basic form for an application using LDAP beans.

1. Instantiate an `LdapConnection` object if you are going to perform any operation on the Directory.
2. Instantiate an `AuthenticateLdap` bean and set its required properties. Naturally, you must authenticate into eDirectory before you can perform any operations on it. If you authenticate without specifying a user, you will be authenticated as a member of the [Public] group, which only has sufficient rights to view the objects in the Directory tree. Any operation attempted will fail because of insufficient rights.
3. Execute the `AuthenticateLdap` bean and catch any exceptions that may occur.
4. Grab the connection you created by assigning the connection to the `LdapConnection` bean you instantiated. This is done by the statement by calling the `AuthenticateLdap` bean's `getLdapConnection()` method and assigning it to the `LdapConnection` bean you created earlier.
5. Now that you have the `LdapConnection` object, you can use this object as an input property to any other LDAP bean you want to instantiate.
6. Unauthenticate from eDirectory using the `UnauthenticateLdap` bean with the `LdapConnection` object as a parameter.

## Simple Authentication Example

Here is a basic servlet demonstrating standard authentication into eDirectory:

---

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.novell.ecb.*;
import com.novell.ecb.ldap.*;

public class BasicAuth extends HttpServlet
{
    private static final String CONTENT_TYPE = "text/html";
    //Initialize global variables
    public void init() throws ServletException
    {
    }
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
```

```

out.println("<font color=\"green\">");
LdapConnection lc = new LdapConnection();
try
{
    AuthenticateLdap auth = new AuthenticateLdap();
    auth.setURL("ldap://edu-qc.provo.novell.com");
    auth.setDN("cn=jnovell, o=novell");
    auth.setPassword("novell");
    auth.execute();
    lc = auth.getLdapConnection();
    out.println("<h1>Welcome to this Basic Authentication Site</h1>");
}
catch(CommandException e)
{
    e.printStackTrace(out);
}
out.println("</font>");
}
public void destroy()
{
}
}

```

Notice I followed the steps outlined previously. Now, if I want to perform any other operations in eDirectory, I have an LdapConnection object with which I can do that. I simply use that as my parameter to another LDAP bean.

## Common Exceptions You Should Handle

The following table lists the common exceptions you should consider handling in your application and the cause of the exception.

Common Exceptions	Cause of the Exception
LdapAuthenticationException	Occurs when a login fails because the password was mis-typed.
CommandException	General exception thrown by a Command Bean. This will probably be the last exception you catch.
LdapException	This is a generic LDAP exception.
LdapCommunicationException	Thrown when the client is unable to communicate via LDAP.
LdapAttributeInUseException	Thrown when an attribute already exists.
LdapInvalidAttributeIdentifier	Thrown when an invalid attribute identifier is used to name an attribute.
LdapInvalidAttributesException	Thrown when an attribute set has been specified incorrectly.
LdapInvalidAttributeValue	Thrown when an attribute value conflicts with the Directory schema.
LdapInvalidNameException	Thrown when the object name is not in correct LDAP form.
LdapInvalidSearchFilter	Thrown when the search filter is incorrect.

Common Exceptions	Cause of the Exception
LdapNameNotFound	Thrown when the object name cannot be found in the Directory tree.
LdapNoPermissionException	Thrown when the client does not have the necessary rights to perform the operation.
LdapNoSuchAttributeDefinition	Thrown when the attribute does not exist in the schema.

## Self-Check

1. eCommerce Beans provide Read only access to eDirectory.
  - a. True
  - b. False
2. Which method clears the input properties of a bean and prepares the bean to be run again?
  - a. clear()
  - b. reset()
  - c. isReady()
  - d. execute()
3. Which method should you call to execute the bean?
  - a. isReady()
  - b. authenticate()
  - c. execute()
  - d. addBean()
4. What should you always do before you run the bean?
  - a. Nothing is required to run the bean
  - b. Authenticate into the Directory
  - c. Reset the bean
  - d. Set the input properties of the bean

5. Which bean would you use to authenticate into the Directory?  
\_\_\_\_\_
6. Which bean would you use to save an authenticated connection to the Directory? \_\_\_\_\_
7. Which bean would you use to unauthenticate from the Directory?  
\_\_\_\_\_
8. Which bean would you use to create a new object in the Directory?  
\_\_\_\_\_
9. Which bean would you use to modify the attributes of an object in the Directory? \_\_\_\_\_
10. Which exception should you catch when the user incorrectly types the password? \_\_\_\_\_
11. Which bean allows you to add a new security provider to your application?  
\_\_\_\_\_
12. Which bean allows you to import a digital certificate into your application?  
\_\_\_\_\_
13. Which certificate format do security beans implement?  
\_\_\_\_\_
14. Which eDirectory service is used to administer cryptographic services in eDirectory? \_\_\_\_\_

- 
- Answers:**
- 1) b. False
  - 2) b. reset()
  - 3) c. execute()
  - 4) d. Set the input properties of the bean
  - 5) AuthenticateLdap
  - 6) LdapConnection
  - 7) UnauthenticateLdap
  - 8) CreatLdapEntry
  - 9) ModifyLdapEntry
  - 10) LdapAuthenticationException
  - 11) AddProvider
  - 12) ImportCertificate
  - 13) x.509
  - 14) Novell Certificate Server

# *Lab Setup*

In the following lab exercises we will build common pieces of a web application and use eDirectory to provide unique authentication and personalization for each user. In order to allow authentication with clear text passwords, you should be sure that clear text authentication is enabled on your server.

## **Enable Clear Text Authentication**

To be sure that clear text authentication is enabled on your server:

1. Open ConsoleOne.
2. Browse to the LDAP Group server object.
3. Right click the object and select properties.
4. On the General tab, be sure that the box is selected to allow clear text passwords.

# Use HTML and a Servlet to Build an Authentication Page

Create an HTML page with two text boxes, one for the username and one for the password. Have the HTML page submit to a Java Servlet that will check to see if the user exists in eDirectory. If the user authenticates, send them to a welcome page that welcomes them to the site and displays their username. If the user does not exist, return a message to the user or redirect them to a page that tells them so. Be sure to catch the necessary exceptions. Remember that the user will not know their LDAP context. The steps to complete the exercise are outlined below.

1. Create an HTML page with a text box and a password field. Add two buttons for submit and reset to the page.
  - a. If you have an HTML editor, you can use it and create a new project with an HTML page called `auth1.html`.
  - b. On the HTML page, insert the doctype declaration first to declare the version of HTML you are using. I have been using XHTML 1.0 so here is my doctype declaration:

---

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

---

- c. Open the HTML tag by typing `<html>`.
- d. Open the HEAD tag by typing `<head>`.
- e. Set a title by typing `<title>Authentication Page</title>`.
- f. Close the HEAD tag by typing `</head>`
- g. Open the BODY tag by typing `<body>`.
- h. Output a header 1 by typing `<h1>Enter your username and password</h1>`.
- i. Open the form tag and complete the method and post attributes of the form. This tells the HTML page where it is posting. The statement is:

---

```
<form method = "post" action = "http://server_name:8080/servlet/auth1">
```

---

- j. Open two input text fields with the input tag and label their names username and password.

---

```
<input type = "text" size = "50" name = "username" />
<input type = "password" size = "50" name = "password" />
```

---

- k.** Open two buttons, a submit and reset button, then close the form, body and html tags.

---

```
<input type = "submit" value = "Authenticate" />
<input type = "reset" value = "Clear Fields" />
```

---

Here is the HTML page I created.

---

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
<head>
<title>Authenticate to eDirectory</title>
</head>

<body bgcolor = "ffffde">

<h1>Please Enter Your Username and Password</h1>

<form method = "post" action =
"http://edu-qc.provo.novell.com:8080/servlet/auth1">

<p><label>Username:<input type = "text" size = "50" name = "username"
/></label></p>
<p><label>Password:<input type = "password" size = "50" name = "password"
/></label></p>
<br>

<p>
  <input type = "submit" value = "Authenticate" />
  <input type = "reset" value = "Clear Fields" />
</p>

</form>

</body>
</html>
```

---

- 2.** Now create a Java Servlet to handle the authentication and dynamic display back to the user whether their authentication was accepted or rejected.

- a.** I created a new project and servlet in JBuilder, which gave me the following stub. We can just add our code to this stub until it is complete.

---

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
```

```

public class Servlet1 extends HttpServlet
{
    private static final String CONTENT_TYPE = "text/html";
    public void init() throws ServletException
    {
    }
    //Process the HTTP Post request
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Servlet1</title></head>");
        out.println("<body bgcolor=\"#ffffff\">");
        out.println("<p>The servlet has received a POST. This is the reply.</p>");
        out.println("</body></html>");
    }
    //Clean up resources
    public void destroy()
    {
    }
}

```

- 
- b.** First you need to add the 3 eCommerce Beans libraries to your classpath and import them into your servlet. The import statements are:

---

```

import com.novell.ecb.*;
import com.novell.ecb.ldap.*;
import com.novell.ecb.security.*

```

---

- c.** Next, you can instantiate an LdapConnection as a class member. We really won't use this object in this program, but we will in the following programs.
- d.** You only need to override the doPost method. So you should put all the code in the doPost method, or if you want to create additional functions outside of doPost and have doPost be the "driver" that works too.
- e.** Instantiate an AuthenticateLdap bean with the line: AuthenticateLdap bean = new AuthenticateLdap();
- f.** Next, create a string that will be used to provide eDirectory with the fully distinguished name and concatenate the LDAP context to the username. I do this with the statement:

---

```
String s = "cn=" + request.getParameter("username") + ", o=novell";
```

---

- g.** Now, set the input properties of the AuthenticateLdap bean with the following statements:

---

```
bean.setURL("ldap://edu-qc.provo.novell.com"); //edu-qc is my
server name. Substitute your server name here.
    bean.setDN(s);
    bean.setPassword(request.getParameter("password"));
```

---

- h.** Now, that the properties are set, you can call the bean's execute method and set your connection object. I also inserted my connection object into the session.
- i.** Now you want to use `out.println()` statements to display a message to the user if authentication succeeded.
- j.** You should catch the following exceptions which will be common errors in the program. Catch `LdapAuthenticationException`, `LdapNameNotFoundException`, `LdapInvalidNameException`, and `CommandException`. Just display a message to the user that the exception occurred.
- k.** You have just written a simple authentication into eDirectory. Take a look at my code for the servlet below.

---

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.novell.ecb.ldap.*;
import com.novell.ecb.*;

public class auth1 extends HttpServlet
{
    private static final String CONTENT_TYPE = "text/html";
    //connection object I will use for any constant connections to the eDirectory
    LdapConnection connection = null;
    public void init() throws ServletException
    {
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        //sets output content type so the servlet knows what to output to the
        browser response.setContentType(CONTENT_TYPE);

        //printwriter is how I output HTML to the browser
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Authentication Servlet</title></head>");
        out.println("<body bgcolor=\"#ffc800\">");

        //Instantiates a new Authentication Bean
        AuthenticateLdap bean = new AuthenticateLdap();

        //puts the username in proper LDAP name
        String s = "cn=" + request.getParameter("username") + ", o=novell";
```

```

//sets the required input properties of the bean
bean.setURL("ldap://edu-qc.provo.novell.com");
bean.setDN(s);
bean.setPassword(request.getParameter("password"));

try
{
    //calls the execute method of the method; the execute method must be
    //contained within a try block. This is where you can catch the necessary
    //exceptions.
    bean.execute();

    //creates a session adds my connection object into the session
    HttpSession session = request.getSession(true);
    connection = bean.getLdapConnection();
    session.setAttribute("Connection", connection);

    if (bean.isAuthenticated())
    {
        //if authentication was successful, I'll let the user know
        out.println("<h1>Welcome, " + request.getParameter("username") +
            "</h1>");
        out.println("<h3>Authentication Succeeded!</h3>");
    }
}
catch(LdapAuthenticationException e)
{
    out.println("<h1>You must have typed your password incorrectly</h1>");
    //for a servlet used in a production environment, I probably would not
    //output the exception to the browser, but just to the console with a
    //System.out.println() statement. I send it to the browser here, just for
    //ease of debugging. the System.out.println() would go to the Tomcat
    //console on the server
    e.printStackTrace(out);
}
catch(LdapNameNotFoundException e)
{
    out.println("<h1>Sorry, your name is not in the Directory!</h1>");
    e.printStackTrace(out);
}
catch(LdapInvalidNameException e)
{
    out.println("<h1>The username is not in the proper LDAP naming
        convention</h1>");
    e.printStackTrace(out);
}
catch(CommandException e)
{
    out.println("<h1>This is a general command bean exception. Check the
        stack trace for more detail.</h1>");
    e.printStackTrace(out);
}
catch (Exception e)
{
    e.printStackTrace(out);
}
out.println("</body></html>");
}
//Clean up resources

```

```
public void destroy()  
{  
}  
}
```

---

# Add Other Methods of Authentication

eCommerce Beans allows you to search through the Directory tree for a specific attribute and then authenticate the user using that attribute. Instead of the username, the client could authenticate with an email address or even another attribute or custom created attribute. Create a second authentication page that will allow the user to authenticate using an attribute other than a username.

1. Create an HTML page with a text field to enter in the email address of a user.
  - a. Open the HTML tag by typing `<html>`.
  - b. Open the HEAD tag by typing `<head>`.
  - c. Set a title by typing `<title>Authentication by Email Address</title>`.
  - d. Close the HEAD tag by typing `</head>`.
  - e. Open the BODY tag by typing `<body>`.
  - f. Output a header 1 by typing `<h1>Enter your email address and password</h1>`.
  - g. Open the form tag and complete the method and post attributes of the form. This tells the HTML page where it is posting. The statement is:

---

```
<form method = "post" action = "http://server_name:8080/servlet/auth2">
```

---

- h. Open one input text field with the input tag and label their names email.

---

```
<input type = "text" size = "50" name = "email" />
```

---

- h.
      - i. Open two buttons, a submit and reset button, then close the form, body and html tags.

---

```
<input type = "submit" value = "Authenticate" />
```

---

---

```
<input type = "reset" value = "Clear Fields" />
```

---

Here is my code for the HTML page.

---

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
<head>
<title>Authentication by Email Address</title>
</head>

<body>

<body bgcolor = "ffffde">

<h1>Please Enter Your Email Address and Password</h1>

<form method = "post" action =
  "http://edu-qc.provo.novell.com:8080/servlet/auth2">

<table>
<tr>
  <td>
    <p>Email Addr:</p>
  </td>
  <td>
    <input type = "text" size = "50" name = "username" />
  </td>
</tr>
<tr>
  <td>
    <p>Password:</p>
  </td>
  <td>
    <input type = "password" size = "50" name = "password" />
  </td>
</tr>
</table>

<br>

<p>
  <input type = "submit" value = "Authenticate" />
  <input type = "reset" value = "Clear Fields" />
</p>

</form>
</body>
</html>

```

---

2. Now, let's create the servlet that will search for the user by his/her email address and lookup the password for the user.
  - a. Begin with the basic servlet stub from the previous exercise.
  - b. Import the ECB libraries with the statements:

---

```
import com.novell.ecb.*;
import com.novell.ecb.ldap.*;
import com.novell.ecb.security.*;
```

---

- c. Instantiate an LdapConnection bean.
- d. In the doPost() method, instantiate a ContextlessLoginLdap bean.
- e. Create a string to pass in the filter method for the email attribute with the statement:

---

```
String s = "mail=" + request.getParameter("email");
```

---

- f. Set the URL, filter, and password properties of the bean with the statements:

---

```
bean.setURL("ldap://edu-qc.provo.novell.com");
bean.setFilter(s);
bean.setPassword(request.getParameter("password"));
```

---

- g. Run the bean's execute() method.
- h. Create an HttpSession object, assign the ldap connection to the LdapConnection bean and add the session to the HttpSession object.

---

```
HttpSession session = request.getSession(true);
connection = bean.getLdapConnection();
session.setAttribute("Connection", connection);
```

---

- i. Welcome the user to the site and output their email address.
- j. Catch LdapAuthenticationException, LdapNameNotFoundException, LdapInvalidNameException, and CommandExceptions and output a message to the user and the stack trace.

You have just written an authentication to eDirectory using an email address. Here is my code.

---

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.novell.ecb.ldap.*;
import com.novell.ecb.*;

public class auth2 extends HttpServlet
{
    private static final String CONTENT_TYPE = "text/html";
    //connection object I will use for any constant connections to the eDirectory
    LdapConnection connection = null;
```

```

public void init() throws ServletException
{
}
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
{
    //sets output content type so the servlet knows what to output to the
    //browser
    response.setContentType(CONTENT_TYPE);

    //printwriter is how I output HTML to the browser
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Authentication Servlet</title></head>");
    out.println("<body bgcolor=\"#ffc800\">");

    //Instantiates a new Authentication Bean
    ContextlessLoginLdap bean = new ContextlessLoginLdap();

    String s = "mail=" + request.getParameter("email");
    System.out.println(s);

    //sets the required input properties of the bean
    bean.setURL("ldap://edu-qc.provo.novell.com");
    bean.setFilter(s);
    bean.setPassword(request.getParameter("password"));

    try
    {
        //calls the execute method of the method; the execute method must be
        //contained within a try block. This is where you can catch the
        //necessary exceptions.
        bean.execute();

        //adds my connection object into the session
        HttpSession session = request.getSession(true);
        connection = bean.getLdapConnection();
        session.setAttribute("Connection", connection);

        if (bean.isAuthenticated())
        {
            //if authentication was successful, I'll let the user know
            out.println("<h1>Welcome, " + request.getParameter("email") +
                "</h1>");
            out.println("<h3>Authentication Succeeded!</h3>");
        }
    }
    catch(LdapAuthenticationException e)
    {
        out.println("<h1>You must have typed your password incorrectly</h1>");
        //for a servlet used in a production environment, I probably would not
        //output the exception to the browser, but just to the console with a
        //System.out.println() statement. I send it to the browser here, just for
        //ease of debugging. the System.out.println() would go to the Tomcat
        //console on the server
        e.printStackTrace(out);
    }
    catch(LdapNameNotFoundException e)
    {
        out.println("<h1>Sorry, your name is not in the Directory!</h1>");
    }
}

```

```
        e.printStackTrace(out);
    }
    catch(LdapInvalidNameException e)
    {
        out.println("<h1>The username is not in the proper LDAP naming
            convention</h1>");
        e.printStackTrace(out);
    }
    catch(CommandException e)
    {
        out.println("<h1>This is a general command bean exception.  Check the
            stack trace for more detail.</h1>");
        e.printStackTrace(out);
    }

    catch (Exception e)
    {
        e.printStackTrace(out);
    }
    out.println("</body></html>");
}
//Clean up resources
public void destroy()
{
}
}
```

---



Here is my code for the HTML page.

---

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
<head>
<title>Authenticate to eDirectory</title>
</head>

<body bgcolor = "ffffde">

<h1>Please Enter Your Username and Password</h1>

<form method = "post" action =
    "http://edu-qc.provo.novell.com:8080/servlet/auth1">

<p><label>Username:<input type = "text" size = "50" name = "username"
/></label></p>
<p><label>Password:<input type = "password" size = "50" name = "password"
/></label></p>

<p><a href = "create.html">Create New User</a></p>

<p>
    <input type = "submit" value = "Authenticate" />
    <input type = "reset" value = "Clear Fields" />
</p>

</form>
</body>
</html>
```

---

2. Now create an HTML page called create.html that the user will use to enter the properties of his user object. This HTML page will post to a servlet which will authenticate into eDirectory and create a user object with the specified username, password and other properties.
  - a. Begin the HTML page with the <html> tag.
  - b. Open the <head> tag.
  - c. Open the <title> tag and type a title for the page such as “Create a User in eDirectory” and close the tag with </title>.
  - d. Close the head tag with </head>.
  - e. Output a header 1 tag with <h1>Enter your username and password</h1>.
  - f. Open the form tag with the method and action properties.
  - g. Open a table tag to format the text fields.
  - h. Open a table row tag by typing <tr>.
  - i. Open a table cell by typing <td>.

- j.** Open a paragraph tag and type: User Name:
- k.** Close the table cell tag.
- l.** Open a new table cell and create a text field with the name username.
- m.** Close the table cell and table row tags.
- n.** Open a new table row and table cell.
- o.** Open a new paragraph tag and type: First Name:
- p.** Close the table cell and table row tags.
- q.** Open a new table cell and row and create a new text field called firstname.
- r.** Close the table cell and row tags.
- s.** Create three more text fields for the last name, email address and password.
- t.** Create submit and rest buttons.
- u.** Close the form, body, and html tags.

Here is my code for create.html:

---

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
<head>
<title>Create a New User Account</title>
</head>

<body bgcolor = "ffffde">

<h1>Enter in the following information to create a user account</h1>

<form method = "post" action =
"http://edu-qc.provo.novell.com:8080/servlet/create">

<table cellpadding = "5" cellspacing = "5">
<tr>
  <td>
    <p>User Name:</p>
  </td>
  <td>
    <input type = "text" size = "50" name = "username" />
  </td>
</tr>
<tr>
  <td>
    <p>First Name:</p>
  </td>
  <td>
```

```

        <input type = "text" size = "50" name = "firstname" />
    </td>
</tr>
<tr>
    <td>
        <p>Last Name:</p>
    </td>
    <td>
        <input type = "text" size = "50" name = "lastname" />
    </td>
</tr>
<tr>
    <td>
        <p>Email Address:</p>
    </td>
    <td>
        <input type = "text" size = "50" name = "email" />
    </td>
</tr>
<tr>
    <td>
        <p>Password:</p>
    </td>
    <td>
        <input type = "password" size = "50" name = "password" />
    </td>
</tr>
</table>

<p>
    <input type = "submit" value = "Authenticate" />
    <input type = "reset" value = "Clear Fields" />
</p>

</form>
</body>
</html>

```

- 
3. Create a servlet that will receive the data from create.html and authenticate to eDirectory as admin and create the user specified in create.html with the properties.
    - a. Begin with the basic servlet stub from Exercise 1.
    - b. First, instantiate an LdapConnection bean.
    - c. Second, instantiate an AuthenticateLdap bean and set the properties so that you log in as admin.
    - d. Now set the Ldap Connection object to the Ldap connection you created by logging in as admin with the AuthenticateLdap bean.
    - e. Now, instantiate a CreateLdapEntry bean passing as a parameter the LdapConnection bean you previously created.
    - f. Create a string and concatenate the proper context with the username the user types in at create.html in the user name text field.

---

```
String s = "cn=" + request.getParameter("username") + ", o=novell";
```

---

- g.** Now, create a string and concatenate the first name and last name together.

---

```
String fullname = request.getParameter("firstname") + " " +  
request.getParameter("lastname");
```

---

- h.** Set each of the properties for the bean with the following statements.

---

```
cle.setName(s);  
cle.addAttribute("objectClass", new String[] {"inetOrgPerson",  
"ndsLoginProperties"});  
cle.addAttribute("uid", request.getParameter("username"));  
cle.addAttribute("sn", request.getParameter("lastname"));  
cle.addAttribute("givenName", request.getParameter("firstname"));  
cle.addAttribute("fullName", fullname);  
cle.addAttribute("userPassword", request.getParameter("password"));  
cle.addAttribute("mail", request.getParameter("email"));
```

---

- i.** If the user is created successfully, output HTML that says that the user was created successfully.

---

```
out.println("<h1 align=center>");  
out.println("User " + request.getParameter("username") + " created  
successfully");  
out.println("</h1>");
```

---

- j.** Catch an `LdapNameAlreadyBoundException` in case the user name specified by the user already exists in eDirectory.

Here is my code for the Create servlet:

---

```
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;  
import java.util.*;  
import com.novell.ecb.*;  
import com.novell.ecb.ldap.*;  
  
public class create extends HttpServlet  
{  
    LdapConnection connection = new LdapConnection();  
    private static final String CONTENT_TYPE = "text/html";  
    public void init() throws ServletException  
    {  
    }  
    //Process the HTTP Post request  
    public void doPost(HttpServletRequest request, HttpServletResponse response)  
        throws ServletException, IOException  
    {
```

---

```

response.setContentType(CONTENT_TYPE);
PrintWriter out = response.getWriter();

out.println("<html>");
out.println("<head><title>create</title></head>");
out.println("<body bgcolor=\`"#ffc800\`">");

//I'll first authenticate as admin
AuthenticateLdap bean = new AuthenticateLdap();
bean.setURL("ldap://edu-qc.provo.novell.com");
bean.setDN("cn=admin, o=novell");
bean.setPassword("password");
try
{
    bean.execute();
}
catch(Exception e)
{
    out.println("<h1>Connection to the Directory failed</h1>");
    e.printStackTrace(out);
}
connection = bean.getLdapConnection();

//The LdapConnection is the parameter to instantiate the CreateLdapEntry bean
CreateLdapEntry cle = new CreateLdapEntry(connection);

//I create a string for the full context of the user object to be created
String s = "cn=" + request.getParameter("username") + ", o=novell";

//I create a string to concatenate the first name and last name to create the
full name attribute
String fullname = request.getParameter("firstname") + " " +
    request.getParameter("lastname");

//I set the attributes for the user object
cle.setName(s);
cle.addAttribute("objectClass", new String[] {"inetOrgPerson",
    "ndsLoginProperties"});
cle.addAttribute("uid", request.getParameter("username"));
cle.addAttribute("sn", request.getParameter("lastname"));
cle.addAttribute("givenName", request.getParameter("firstname"));
cle.addAttribute("fullName", fullname);
cle.addAttribute("userPassword", request.getParameter("password"));
cle.addAttribute("mail", request.getParameter("email"));

try
{
    cle.execute();
    out.println("<h1 align=center>");
    out.println("User " + request.getParameter("username") + " created
        successfully");
    out.println("</h1>");
}
catch(LdapNameAlreadyBoundException e)
{
    //Catch this exception if the user already exists in eDirectory
    out.println("<h1>The user name you have chosen already exists</h1>");
    e.printStackTrace(out);
}
catch(Exception e)

```

```
{
    e.printStackTrace(out);
}
    out.println("</body></html>");
}
//Clean up resources
public void destroy()
{
}
}
```

---

# Allow the User to Change Attributes in the Directory

Create a page that will display text boxes for associated attributes in the Directory. The user should be allowed to enter the necessary attributes and submit them. On submit, the attributes should be written into the Directory.

1. Create an HTML page with text fields to modify each of the attributes for a user object in eDirectory.
  - a. I used the same HTML file from the previous exercise. Here is the code from my file.

---

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<title>Create a New User Account</title>
</head>

<body bgcolor = "ffffde">

<h1>Enter in the following information to modify the attributes</h1>

<form method = "post" action =
"http://edu-qc.provo.novell.com:8080/servlet/create">

<table cellpadding = "5" cellspacing = "5">
<tr>
  <td>
    <p>User Name:</p>
  </td>
  <td>
    <input type = "text" size = "50" name = "username" />
  </td>
</tr>
<tr>
  <td>
    <p>First Name:</p>
  </td>
  <td>
    <input type = "text" size = "50" name = "firstname" />
  </td>
</tr>
<tr>
  <td>
    <p>Last Name:</p>
  </td>
  <td>
    <input type = "text" size = "50" name = "lastname" />
  </td>
</tr>
```

```

        </td>
</tr>
<tr>
    <td>
        <p>Email Address:</p>
    </td>
    <td>
        <input type = "text" size = "50" name = "email" />
    </td>
</tr>
<tr>
    <td>
        <p>Password:</p>
    </td>
    <td>
        <input type = "password" size = "50" name = "password" />
    </td>
</tr>
</table>

<p>
    <input type = "submit" value = "Authenticate" />
    <input type = "reset" value = "Clear Fields" />
</p>

</form>
</body>
</html>

```

- 
2. Create a servlet to receive the attributes from the HTML page and change them in eDirectory.
    - a. Create an LdapConnection object.
    - b. Create an AuthenticateLdap object and log in as admin into the eDirectory tree.
    - c. Assign the LdapConnection to the LdapConnection of the AuthenticateLdap bean.
    - d. Instantiate a ModifyLdapEntry bean.
    - e. Create a string and concatenate the username with the user's context.
    - f. Set the name of the ModifyLdapEntry bean.
    - g. Replace the other attributes that the user specified with the following lines.

---

```

mle.replaceAttribute("givenName", request.getParameter("firstname"));
mle.replaceAttribute("sn", request.getParameter("lastname"));
mle.replaceAttribute("mail", request.getParameter("email"));
mle.replaceAttribute("fullName", request.getParameter("firstname") + " " +
request.getParameter("lastname"));

```

---

- h.** Execute the bean and let the user know that the modifications were made successfully.
- i.** Instantiate a ChangePasswordLdapEntry bean.
- j.** Set the user name with the user name provided from the HTML page.
- k.** Set the password from the HTML page.
- l.** Execute the bean and let the user know if the modifications were made successfully.

Here is the code from the modify servlet.

---

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.novell.ecb.*;
import com.novell.ecb.ldap.*;

public class modify extends HttpServlet
{
    LdapConnection connection = null;
    LdapConnection connection2 = null;
    private static final String CONTENT_TYPE = "text/html";
    public void init() throws ServletException
    {
    }
    //Process the HTTP Post request
    public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException
    {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head><title>modify</title></head>");
        out.println("<body bgcolor=\"#ffc800\">");

        //I'll first authenticate as admin
        AuthenticateLdap bean = new AuthenticateLdap();
        bean.setURL("ldap://edu-qc.provo.novell.com");
        bean.setDN("cn=admin, o=novell");
        bean.setPassword("password");
        try
        {
            bean.execute();
        }
        catch(Exception e)
        {
            out.println("<h1>Connection to the Directory failed</h1>");
            e.printStackTrace(out);
        }
        connection = bean.getLdapConnection();
        connection2 = bean.getLdapConnection();

        ModifyLdapEntry mle = new ModifyLdapEntry(connection);
```

```

String s = "cn=" + request.getParameter("username") + ", o=novell";
mle.setName(s);
mle.replaceAttribute("givenName", request.getParameter("firstname"));
mle.replaceAttribute("sn", request.getParameter("lastname"));
mle.replaceAttribute("mail", request.getParameter("email"));
mle.replaceAttribute("fullName", request.getParameter("firstname") + " "
    + request.getParameter("lastname"));
try
{
    mle.execute();
    out.println("<h1>User: " + request.getParameter("username") + "
        modified successfully</h1>");
}
catch(Exception e)
{
    e.printStackTrace(out);
}

SetPasswordLdapEntry sple = new SetPasswordLdapEntry(connection2);
sple.setName(s);
sple.setPassword(request.getParameter("password"));
try
{
    sple.execute();
    out.println("<h1>Password for user: " +
        request.getParameter("username") + " changed successfully</h1>");
}
catch(Exception e)
{
    e.printStackTrace(out);
}
out.println("</body></html>");
}
//Clean up resources
public void destroy()
{
}
}

```

---

# Display the Attributes for the User to See

Create a page that will query the Directory and return to the user the attribute name and value. Link this page together with an authentication page and the modify attribute page. The user should authenticate into the Directory and see a page prompting the user to view his/her profile or edit his/her profile.

1. Create an HTML page with a text box that will accept a username and post the username to a servlet.
  - a. Open the <html> tag.
  - b. Open the <head> tag.
  - c. Open the <title> tag and type View Properties of a User and close the title tag.
  - d. Close the </head> tag.
  - e. Open the <body> tag.
  - f. Output an <h1> tag and type Enter the username to view its properties.
  - g. Close the </h1> tag.
  - h. Open the form tag with the method and action attributes.

---

```
<form method = "get" action = "http://edu-qc.provo.novell.com:8080/servlet/view" >
```

---

- i. Create a text box with the name "username".
- j. Create submit and reset buttons.
- k. Close the </form> tag.
- l. Close the </body> and </html> tags.

Here is the code for my HTML page view.html.

---

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
<head>
<title>View Properties of a User</title>
</head>
```

```

<body bgcolor="#ffc800">
<h1>Enter the username to view its properties</h1>

<form method = "get" action = "http://edu-gc.provo.novell.com:8080/servlet/view"
>

<p>
  <label>Username: <input type = "text" name="username" size = "50" /></label>
</p>

<p>
<input type="submit" value="Submit" />
<input type="reset" value="Reset" />
</p>

</form>
<hr>
</body>
</html>

```

2. Create a servlet that overrides the doGet() method and queries a user object for its properties and displays the properties and values in an HTML table.
  - a. Instantiate an LdapConnection bean.
  - b. Instantiate an AuthenticateLdap bean and login as admin to the server.
  - c. Assign the LdapConnection object to the LDAP connection you created with the Authenticate bean.
  - d. Instantiate a ReadLdapEntry bean.
  - e. Create a string and concatenate the context with the user's common name.
  - f. Set the name of the ReadLdapEntry bean with the setName() method.
  - g. Execute the bean.
  - h. Output HTML statements to create a table with two columns and as many rows as there are attributes for the object in eDirectory.
  - i. Type out.println("<table>");
  - j. Type  
out.println("<thead><th>Attributes</th><th>Values</th></thead>");
  - k. Create a for loop that will loop through each attribute, call its value and output it to the screen with HTML.

```

for (int i = 0; i < rle.getAttributeCount(); i++)
{
  out.println("<tr><td>");
  out.println(a[i]);
}

```

```

        out.println("</td><td>");
        out.println(rlc.getAttribute(a[i]) + "</td></tr>");
    }
    out.println("</table>");

```

---

- I. Catch an `LdapNameNotFoundException` and let the user know that the user object could not be found in eDirectory.

Here is my code for the View servlet.

---

```

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.novell.ecb.*;
import com.novell.ecb.ldap.*;

public class view extends HttpServlet
{
    LdapConnection connection = null;
    private static final String CONTENT_TYPE = "text/html";

    public void init() throws ServletException
    {
    }
    //Process the HTTP Get request
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head><title>View</title></head>");
        out.println("<body bgcolor=\`#\`ffc800\`>");

        AuthenticateLdap bean = new AuthenticateLdap();
        bean.setURL("ldap://edu-qc.provo.novell.com");
        bean.setDN("cn=admin, o=novell");
        bean.setPassword("password");

        try
        {
            bean.execute();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        connection = bean.getLdapConnection();

        ReadLdapEntry rlc = new ReadLdapEntry(connection);
        String s = "cn=" + request.getParameter("username") + ", o=novell";
        rlc.setName(s);
        try
        {
            rlc.execute();
            out.println("<h3># of Attributes = " + rlc.getAttributeCount() +

```

```

        "</h3><br>");
String[] a = rle.getEntryAttributeNames();
out.println("<table border=1 cellpadding=5>");
out.println("<thead><th>Attributes</th><th>Values</th></thead>");
for (int i = 0; i < rle.getAttributeCount(); i++)
{
    out.println("<tr><td>");
    out.println(a[i]);
    out.println("</td><td>");
    out.println(rle.getAttribute(a[i]) + "</td></tr>");
}
out.println("</table>");
}
catch(LdapNameNotFoundException e)
{
    out.println("<h1>Sorry, eDirectory did not find the name you
        requested.</h1>");
    e.printStackTrace(out);
}
catch(Exception e)
{
    e.printStackTrace(out);
}
out.println("</body></html>");
}
//Clean up resources
public void destroy()
{
}
}
}

```

---

# *Establish Secure Connection and Unauthenticate from eDirectory*

Your user would never enter information over the Internet without using a secure connection. Change the Modify servlet to use a SSL connection and Unauthenticate from the Directory.

1. Use the HTML page from Exercise #4. Assign rights to the page so that the page is accessed from a secure connection, for example https instead of http. Check the documentation of your Web server for instructions on how to do this.
2. Use the same Java code as Exercise #4. Add the functionality to use a secure connection and authenticate from the Directory.
  - a. Before you execute the AuthenticateLdap bean, add the line to use the SSL protocol by typing: `bean.setProtocol("SSL");`
  - b. Change the URL of your server to use port 636 instead of port 389, which is the default LDAP port if one is not specified.
  - c. After you assign the LdapConnection to the connection you created with the AuthenticateLdap bean, output which protocol you are using by typing: `connection.getProtocol();`
  - d. After you execute the ModifyEntry bean, instantiate an UnauthenticateLdap bean passing to it as a parameter the LdapConnection object.
  - e. Execute the bean.
  - f. Output HTML letting the user know that the connection has been unauthenticated with the statement:

---

```
out.println("<h1>Authenticated: = " + u.isAuthenticated() + "</h1>");
```

---

Here is my code for the servlet.

---

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import com.novell.ecb.*;
import com.novell.ecb.ldap.*;
```

```

public class modify extends HttpServlet
{
    LdapConnection connection = null;
    private static final String CONTENT_TYPE = "text/html";
    public void init() throws ServletException
    {
    }
    //Process the HTTP Post request
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<head><title>modify</title></head>");
        out.println("<body bgcolor=\\"#ffc800\>");

        //I'll first authenticate as admin
        AuthenticateLdap bean = new AuthenticateLdap();
        bean.setURL("ldap://edu-qc.provo.novell.com:636");
        bean.setDN("cn=admin, o=novell");
        bean.setPassword("password");
        bean.setProtocol("ssl");
        try
        {
            bean.execute();
        }
        catch(Exception e)
        {
            out.println("<h1>Connection to the Directory failed</h1>");
            e.printStackTrace(out);
        }
        connection = bean.getLdapConnection();
        out.println("<h1>Connection Protocol is: " + connection.getProtocol() +
            "</h1>");

        ModifyLdapEntry mle = new ModifyLdapEntry(connection);
        String s = "cn=" + request.getParameter("username") + ", o=novell";
        mle.setName(s);
        mle.replaceAttribute("givenName", request.getParameter("firstname"));
        mle.replaceAttribute("sn", request.getParameter("lastname"));
        mle.replaceAttribute("mail", request.getParameter("email"));
        mle.replaceAttribute("fullName", request.getParameter("firstname") + " "
            + request.getParameter("lastname"));
        try
        {
            mle.execute();
            out.println("<h1>User: " + request.getParameter("username") + "
                modified successfully</h1>");
        }
        catch(Exception e)
        {
            e.printStackTrace(out);
        }

        SetPasswordLdapEntry sple = new SetPasswordLdapEntry(connection);
        sple.setName(s);
        sple.setPassword(request.getParameter("password"));
    }
}

```

```

try
{
    sple.execute();
    out.println("<h1>Password for user: " +
        request.getParameter("username")+ " changed successfully</h1>");
}
catch(Exception e)
{
    e.printStackTrace(out);
}

UnauthenticateLdap u = new UnauthenticateLdap(connection);

try
{
    u.execute();
    out.println("<h1>Authenticated = " + u.isAuthenticated() + "</h1>");
}
catch(Exception e)
{
    out.println("Error Unauthenticating...");
    e.printStackTrace();
}
out.println("</body></html>");
}
//Clean up resources
public void destroy()
{
}
}

```

---

# Security Beans

In order to understand the functionality of eCommerce Security Beans, it is necessary to discuss the Java Security model and related concepts.

Java Security Terms	Explanation
Security Domain	In Java terms, a logical group where the Java program runs. The Internet could be considered one domain. The smaller domain could be the local computer. If domains are not trusted, then you should use a secure connection.
Digital Certificate	A digital "signature" that verifies the program originates from a specific company and that the program is not harmful to your system.
Security Algorithm	An algorithm such as RSA that is used to encrypt information sent between two domains.
Cryptographic Service Provider	A package that allows the programmer to implement a security algorithm.
KeyStore	A secure database that stores digital certificates.

## Java Security Model

The Java Cryptographic Architecture (JCA) is the API that contains the Java Security packages. The Java Cryptographic Extensions (JCE) is an extension of the JCA that includes encryption and key exchange.

### Novell Implementation

eCommerce Security Beans contain the functionality to implement Java security into your Web application. Similarly to LDAP Beans, Security Beans are written entirely in Java and contain within them one logic tasks per bean. This allows you to easily integrate the beans in your Web application.

Security Beans allow you to create your own security provider, digital certificates, and retrieve certificates from remote hosts. This allows users to verify the authenticity of your application.

Novell uses open standards in the implementation of Security Beans. X.509 certificates, which are certificates that provide a standard way of storing and retrieving the information on the certificates, have been implemented in the certificate class.

You have a choice of encryption algorithms to implement or a combination of the security algorithms.

## Novell Certificate Server

Novell Certificate Server can be an integral part of your application's security implementation. Certificate Server uses eDirectory to store the Organizational Authority, KeyStore, and host the PKI services you need. The best part is that it is all in eDirectory. You can use eCommerce Security Beans to query eDirectory for the certificates for your application. Also with Certificate Server, you have a central administration point of your PKI services. Since Certificate Server details are beyond the scope of this course, see <http://www.novell.com/documentation> for more details.

Commonly Used Security Beans	Explanation
AddProvider	Command Bean that adds a security provider to the java.security file.
RemoveProvider	Command Bean that removes a security provider from the java.security file.
ListProvider	Command Bean that lists the current security providers.
ImportCertificate	Imports a digital certificate into a KeyStore.
ListCertificates	Lists the certificates in a given KeyStore.
RetrieveHostCertificates	Retrieves certificates from a remote host.
DeleteCertificate	Deletes a certificate from a KeyStore.

# *Create an eCommerce Application (Optional Lab)*

Create an online shopping Web site. Store the products and product information in eDirectory. You can use sessions to create the shopping cart. A combination of JSP and Servlets would best fit this exercise.

# Conclusion

You have reached the end of the Novell eCommerce Beans course.

## Checking for Success

During this course you have become familiar with Novell's eCommerce Beans and have gained some exposure to using these beans to integrating Web applications with eDirectory by building a calendar application. As you performed these tasks, you learned the following concepts and skills:

- What is a Directory?
- Novell eCommerce Beans Overview
- Sessions
- Novell eCommerce Beans Introduction
- LDAP Beans
- Lab Setup
- Use HTML and a Servlet to Build an Authentication Page
- Add Other Methods of Authentication
- Add a Link to Create a New User
- Allow the User to Change Attributes in the Directory
- Display the Attributes for the User to See
- Establish Secure Connection and Unauthenticate from eDirectory
- Security Beans
- Create an eCommerce Application (Optional Lab)

You are now familiar enough with Novell eCommerce Beans to create your own Web application. With the information you've learned you can leverage Novell eCommerce Beans to integrate your LDAP Directory into a Web application. Feel free to return to this course any time to brush up your skills.

## Additional Resources

For more information, check out the following resources:

- <http://www.ldapzone.com>
- <http://www.developer.novell.com>
- <http://java.sun.com>

To find out about other Developer Net University courses that are available to you, visit <http://www.developer.novell.com/education>.

To take advantage of all the other technical knowledge available to you visit AppNotes online (or find out how you can subscribe to the hard copy publication) at <http://developer.novell.com/research/>.

Copyright © 2002 by Novell, Inc. All rights reserved.  
No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Novell.

All product names mentioned are trademarks of their respective companies or distributors.