developerWorks

LDD Today

Lotus. software

Building real-time access to an LDAP directory server from your Notes application

Level: Intermediate
Works with: Notes/Domino
Updated: 22-Sep-2003

by
Aditya T.
Wresniyandaka

With today's growing demand for centralized, enterprise-wide directory services, you may soon have to integrate your Domino application with your LDAP directory server. In your Domino application, you may want to have fields on a Notes form automatically populated with information from an LDAP directory server. One option for achieving this integration is to write a Java agent that connects to your enterprise directory server using the Java Naming and Directory Interface (JNDI).

JNDI is a standard extension to the Java platform that provides connectivity to enterprise naming and directory services, such as Lightweight Directory Access Protocol (LDAP). Java applications, including Domino Java agents, can benefit from this extension because it is independent of a particular directory or naming service implementation. You are probably familiar with the procedure for setting up directory assistance for users who want to authenticate to your Domino server using credentials stored in an external LDAP server.

This article shows you how to prepare your Domino environment and explains the important building blocks for your Java agent. The environment preparation is not necessary if you are running Notes/Domino 6 because the required JAR files are already included on the client and server. You will see several test scenarios using a Notes form that passes parameters to this agent. For example, you can search for a single value of an attribute, such as phone number, or multiple values, such as members in a department. To try the example described in this article, you do *not* need a Directory Assistance document on your Domino server. The directory service API handles the connection to your external LDAP server. This does not necessarily mean that you can eliminate the need for a Directory Assistance database, especially if you have multiple Domino directories or if you have Domino applications that require authentication to an external directory server.

This article assumes that you are an experienced Domino developer with a Domino server administration background and that you are familiar with LDAP. For more in-depth discussion about JNDI, see the JNDI Tutorial Web site, and for detailed information on how to pass parameters to an agent, see the documentation in the Agent FAQ in the Notes/Domino 4 and 5 Forum.

Environment

The design elements described in this article were developed and tested using the following:

- Lotus Domino 5.0.9a and later on Windows 2000
- Lotus Domino Designer 5.0.9a and later on Windows 2000
- Lotus Notes client 5.0.9a and later on Windows 2000
- Directory Server: OS/400 V5R1 Directory Services (option 32)

System preparation
This section applies if you are running an R5 environment. If your client and server run Notes/Domino 6, you can skip this section and go directly to the Application design section.

General
The first step is to obtain the JNDI packages. These packages are not part of the Domino server, so you must download them from the JNDI Web site. Make sure that you download JNDI 1.1.2 because Domino R5 supports JDK 1.1.8. The components that you use in your application are JNDI 1.1.2 Class Libraries (jndi112.zip) and LDAP Service Provider 1.0.3 (ldap103.zip).

Follow the instructions to download and extract the following JAR files:
- Jndi.jar
- Ldap.jar
- Providerutil.jar (Note that this JAR file exists in more than one zip file, so it doesn't matter where you get the JAR file from.)

Next, you need to include these JAR files in your Notes/Domino environment. You have two options: The first is to include or attach these JAR files to your Java agent. This way, when your application replicates to multiple servers, you have the correct JAR files needed by your application. The second option is to specify these JAR files in your server's Notes.ini file using the JavaUserClasses parameter. This is practical if the JAR files are used or shared by multiple applications. When you need to update the JAR files, you only need to update them in one place. In the next section, we describe the second option in more detail.

The Domino Designer
Update your Notes.ini file to include these JAR files in the JavaUserClasses parameter, for example:

```
JavaUserClasses=c:\MyJarFiles\lib\jndi.jar;c:\MyJarFiles\lib\ldap.jar;c:\MyJarFiles\lib\providerutil.jar.
```

Save the Notes.ini file and restart Domino Designer.

The Domino server
Now it's time to include the JAR files in your Domino server environment. Follow these steps:
1. Copy the JAR files to your Domino server. If you have Windows 2000, copy the three JAR files to the Java directory on your server. If you have an iSeries (OS/400) server, copy these JAR files to an iSeries IFS directory using FTP with the binary option or mapping a local drive on your PC to the Domino\data directory on your iSeries server. Change the ownership of these files to QNOTES, and update the authority for these objects by adding QSYS with *RWX access.
2. Update your Domino server Notes.ini file by adding the JavaUserClasses parameter and its value (see example for Domino Designer). On the iSeries, use a colon as a separator between the jar files, for example:

   ```
   JavaUserClasses=/home/myjarfiles/lib/jndi.jar:/home/myjarfiles/lib/ldap.jar:
   /home/myjarfiles/lib/providerutil.jar
   ```

3. Restart your Domino server. You should see the completion message:

   ```
   JVM: Java Virtual Machine initialized
   ```

However, you may also receive the following error messages:

```
Addin: Agent error message: java.lang.ClassFormatError: com/sun/jndi/ldap/LdapCtxFactory
```

This may be an indication that the JAR files are not compatible with the JDK level supported by your Domino server. If you have iSeries, you could receive the following error message:

```
Addin: Agent error message: Attaching Java program to /home/myjarfiles/lib/jndi.JAR
```

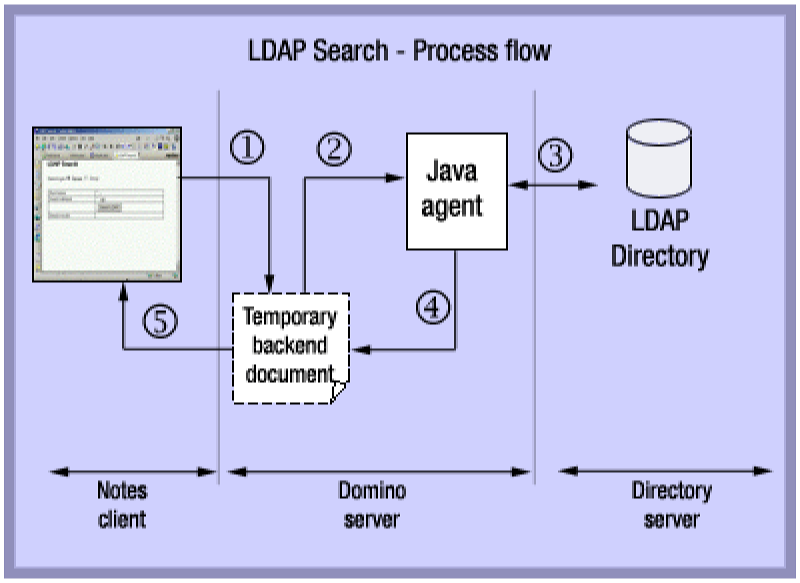
This error message appears if the QSYS on iSeries is not given authority to the JAR files.

If your Domino Designer and Domino server start successfully, you are now ready to write your application.

Application design

The following procedure describes how the application that we develop to connect with the LDAP directory server works:

1. You perform a search using the LDAP Search form. This form has several fields in it. First, you choose Person or Group. If you choose Person, you type the name, select an attribute from the list, and click the Search LDAP button. If you choose Group, you type the group name and exit the field.
2. The Search LDAP button or the Exiting event saves the form to a back-end document and calls the Java agent, LDAPSearchWithFilter, by passing the doc.NoteID value as a parameter.
3. The agent connects to the LDAP server and performs the search.
4. The agent updates the back-end document with the search results (actual data or error messages). For simplicity, the assumption used in the code is that only one entry is returned at a time. However, each entry can have more than one attribute value.
5. The LDAP Search form reopens the back-end document by its doc.NoteID value and performs the necessary string manipulations before displaying the results.



The Java agent

The Java agent runs on the server so that you do not have to distribute and install the JAR files on each client. When you create the agent, these are the properties that you specify:

- When should this agent run: Manually from actions menu
- Which document(s) should it act on: Run once (@Commands may be used)
- Type: Java

Select the Shared option, and before you run the agent on the server, make sure that it is signed properly.

The complete code for the LDAPSearchWithFilter agent is available in this sidebar. The agent consists of several areas or blocks of code. The import section and main body of the agent are as follow:

```
import lotus.domino.*;
import javax.naming.*;
import javax.naming.directory.*;
import java.util.Hashtable;
import java.util.Vector;
```

In the previous code snippet, javax.naming is the package that contains classes and interfaces for the naming operation. Javax.naming.directory is the extension of javax.naming. It provides functionality for accessing directories. A hash table and a vector are used to populate the JNDI Context and the search results respectively. The use of a vector gives you an advantage that it can hold a single value (if you search for a person's attribute) or multiple values (if you search for members of a group) dynamically.

```
public class LDAPSearchWithFilter extends AgentBase {
    public void NotesMain() {
        try {
            Database _db;
            Document _doc;
            Session session = getSession();
            AgentContext agentContext = session.getAgentContext();

            _db = agentContext.getCurrentDatabase();
```

NotesMain is the main body of the agent that performs the tasks described in the following sections.

Retrieve parameters from the Notes document
The fields that are used for search are searchCN and searchAttr stored in the back-end document, identified by the doc.NoteID value:

```
    Agent ag1 = agentContext.getCurrentAgent();
    String paramid = ag1.getParameterDocID();
    Document doc = _db.getDocumentByID(paramid);

    String searchCN = doc.getItemValueString("SearchCN");
    String searchAttr = doc.getItemValueString("SearchAttr");
```

Create the JNDI context factory
Here you specify the address and port of your LDAP server, the base DN for search, and the ID and password for connecting to your LDAP server. Note that you can also store this information in a Keyword document. In this example, you use an anonymous connection.

```
    Hashtable env = new Hashtable(11);
    env.put(Context.INITIAL_CONTEXT_FACTORY, com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, "ldap://myldap.mycompany.com:389/ou=myou, o=myorg, c=us");
    If your Directory server requires authentication, add the following lines in your code (check with your Directory
    server administrator for the correct ID and password):
    env.put(Context.SECURITY_PRINCIPAL, “cn=Administrator”);
    env.put(Context.SECURITY_CREDENTIALS, “thepassword”);
```

Define the methods
The DirContext interface defines the methods for examining and updating the attributes associated with directory objects, as well as the methods for searching the directory itself.

```
    DirContext ctx = new InitialDirContext(env);
```

Define search filter and attributes
The SearchControls object defines the search filter and the attributes that need to be returned. The default is to search one level, but you can also specify search on the entire subtree. (Refer to the JNDI documentation if you want to set a different search scope in the SearchControls class.) The search operation returns the results to the NamingEnumeration object.

```
    String[] attrIDs = {searchAttr};
    SearchControls ctls = new SearchControls();
    ctls.setReturningAttributes(attrIDs);
    // Specify the search filter
    String filter = "(|(cn="+searchCN+")(uid="+searchCN+"))";

    // Search for objects using the above filter
    NamingEnumeration answer = ctx.search("", filter, ctls);
```

Use the findCN method to determine if an object exists
This method determines if an LDAP entry or object being searched, based on the content of answer, exists. If it does, the method calls the findAttributes method. If not, it updates the hidden field HiddenSearchResults with the value CN not found.

```
public static void findCN(NamingEnumeration enum, Document doc) {
    try {
        if (enum.hasMore()) {
        while (enum.hasMore()) {
        SearchResult sr = (SearchResult)enum.next();
        findAttrs(sr.getAttributes(), doc);
    }
    } else {
        Item item = doc.getFirstItem("HiddenSearchResults");
        doc.replaceItemValue("HiddenSearchResults", "CN not found");
        doc.save(true);
    }
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

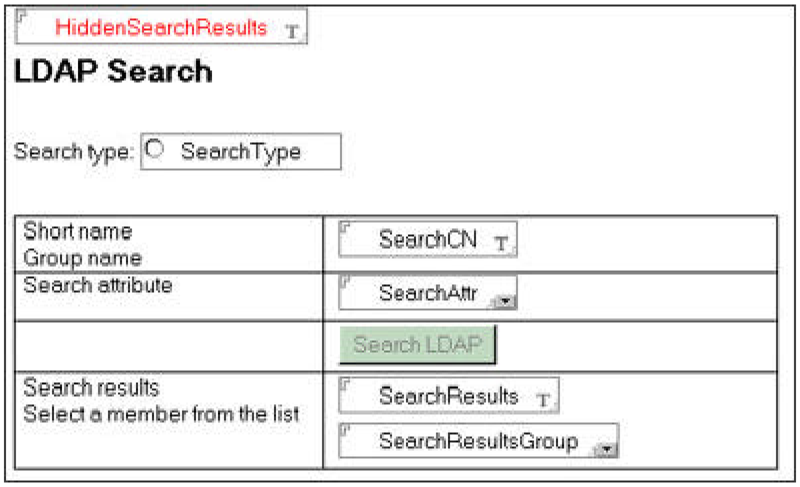
Use the findAttrs method to locate attributes and values
This method traverses the directory server entries for the attribute and its values. Keep in mind that an attribute can have multiple values. If found, these values are stored in a vector, which can be used to update the hidden field HiddenSearchResults. At the end of the process, it saves the back-end document. The back-end document is later retrieved and reopened by the front-end (UI) document.

```
public static void findAttributes(Attributes attrs, Document doc) {
    try {
        Item item = doc.getFirstItem("HiddenSearchResults");
        Vector v = new Vector();
        String result;

    if (attrs.size() == 0) {
        v.addElement("Name found but attribute is blank or not found. Try again …");
    } else {
    /* Get each attribute */
        try {
            for (NamingEnumeration ae = attrs.getAll(); ae.hasMore();) {
            Attribute attr = (Attribute)ae.next();
            /* Get each value */
            for (NamingEnumeration e = attr.getAll(); e.hasMore();) {
                result = (String)e.next();
                v.addElement(result);
            }
        }
        } catch (NamingException e) {
            e.printStackTrace();
        }
    } // end if
        doc.replaceItemValue("HiddenSearchResults", v);
        doc.save(true);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

The search form

The LDAP Search form has several fields for entering or selecting the search parameters and for holding and displaying the results. The following screen shows the LDAP Search form.



The following table describes the fields on this form:

Field	Description
HiddenSearchResults	A hidden, editable text field.
SearchType	A radio-button field; the values are Person or Group.
SearchCN	An editable text field for inputting your search query; the Entering event in this field clears the other fields when you are about to type a search word. The Exiting event runs the LDAPSearchWithFilter agent, if the value in the SearchType field is Group. Code examples for the Entering and Exiting events can be found in this sidebar.
SearchAttr	A dialog list field; in this example, you will populate this field with the following choices: Last name | sn Email | mail Phone | telephoneNumber Sn, mail, and telephoneNumber are the LDAP attributes that you can search. Note that you will not see this field if the search type is Group.
Search LDAP	A button that calls the SearchLDAPWithFilter agent, using LotusScript; the button is hidden if the SearchType is Group. Code for the Search LDAP button is available in this sidebar.
SearchResults and SearchResultsGroup	Computed text fields; the default values are SearchResults and SearchResultsGroup respectively.

There are two types of search that you can perform using this form:

- *An attribute that belongs to a person; SearchType = "Person"*
  The user types the name and selects an attribute from the list. The Search LDAP button calls the LDAPSearchWithFilter agent and presents the result. For an individual attribute search, the result is displayed using the SearchResults field, whose value is retrieved from the HiddenSearchResults field.
- *Members in a group; SearchType = "Group"*
  The user types the name of the group, and upon exiting the field, the LDAPSearchWithFilter agent is called. You use the results to populate the SearchResultsGroup field. If the search does not return any results, the Group not found message is displayed in this field.

For a group search, the agent returns the names in an LDAP format, for example:

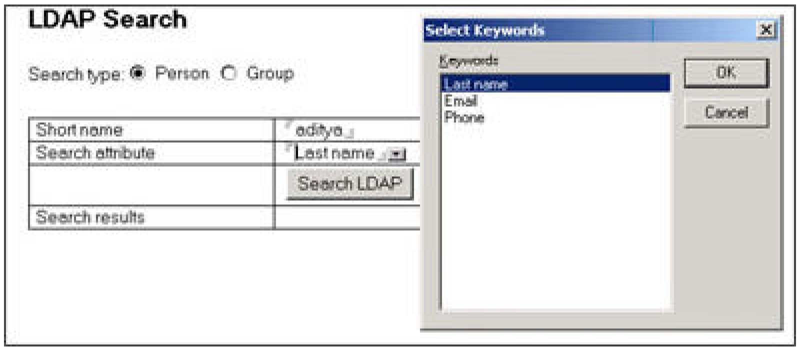
```
cn=John Doe, ou=myou, o=myorg, c=us
```

After retrieving and reopening the back-end document, you need to extract the cn and to store the value in the HiddenSearchResults field. Here is a code snippet that extracts the value John Doe from the above example:

```
Set searchResultItem = doc.getFirstItem("HiddenSearchResults")
Forall values In searchResultItem.Values
   pos1 = Instr(Cstr(values), "=")
   pos2 = Instr(Cstr(values), ",")
   stringValue =Mid(Cstr(values), pos1+1, pos2-pos1)
   Call uidoc.FieldAppendText("HiddenSearchResults", stringValue)
```

Testing your application

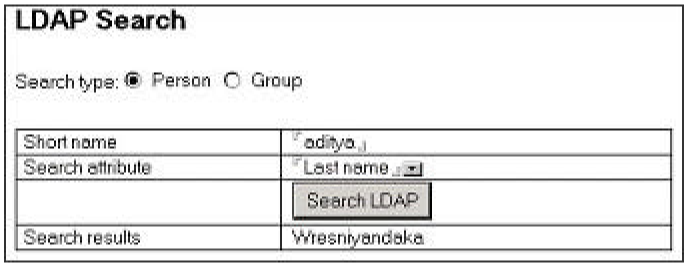
You are now ready to test your application. In the first test scenario, an entry exists on an external directory server (which is running on OS/400). The person has last name and email address, but no phone number, so you search by last name. The following screen shows a search by last name.


LDAP Search

Search type: (●) Person ( ) Group

Short name	aditya
Search attribute	Last name
	Search LDAP
Search results	

The following screen shows the last name search results.

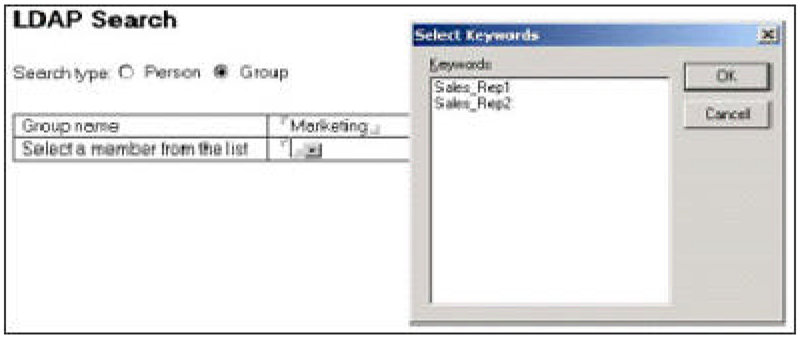

LDAP Search

Search type: (●) Person ( ) Group

Short name	aditya
Search attribute	Last name
	Search LDAP
Search results	Wresniyandaka

If the name (cn), attribute, or both are not found, you receive an error message. For example, if you change the search attribute to phone, you receive an error: "Name found but attribute is blank or not found. Try again."

In the second test scenario, you generate a list of all members of a group. For example, if you search for members in the Marketing group, you receive a dialog list of group members.


LDAP Search

Search type ( ) Person (•) Group

Group name	Marketing
Select a member from the list	

If the group you are searching for does not exist, then you receive a Group not found message in the dialog box in place of the group member names.

Conclusion

With the availability of the JNDI API, you can now integrate your Notes application with your enterprise directory server in a real-time fashion.You do not have to make a copy of information from your directory server to be used in your Notes application. This approach eliminates the issues related to data currency.

ABOUT THE AUTHOR

Aditya Wresniyandaka is a certified Lotus Notes/Domino, WebSphere Application Server, and e-business specialist at Rochester iSeries Services Group (IBM Global Services) in Rochester, MN. He has been working with iSeries technology since 1990, currently focusing on Lotus Notes/Domino and WebSphere Application Server. Aditya has an MS in Management Information Systems from the University of Arizona in Tucson, AZ. In his spare time, he enjoys having fun with his digital camera and photo editing projects.

The LDAPSearchWithFilter agent code

The following is the complete code example for the LDAPSearchWithFilter agent.

```
import lotus.domino.*;
import javax.naming.*;
import javax.naming.directory.*;
import java.util.Hashtable;
import java.util.Vector;

public class LDAPSearchWithFilter extends AgentBase {

   public void NotesMain() {

      try {
         Database _db;
         Document _doc;
         Session session = getSession();
         AgentContext agentContext = session.getAgentContext();

         _db = agentContext.getCurrentDatabase();

         // Used for running agent on server with docID being passed from the calling action
         Agent ag1 = agentContext.getCurrentAgent();
         String paramid = ag1.getParameterDocID();
         Document doc = _db.getDocumentByID(paramid);

         String searchCN = doc.getItemValueString("SearchCN");
         String searchAttr = doc.getItemValueString("SearchAttr");

         // Set up the environment for creating the initial context
         String ldapCF = "com.sun.jndi.ldap.LdapCtxFactory";
         String ldapURL = "ldap://myldap.mycompany.com:389/";
         String ldapBaseDN = "ou=myou, o=myorg, c=us";
         String ldapUserID = "cn=Administrator";
         String ldapPassword = "thepassword";

         Hashtable env = new Hashtable(4);
         env.put(Context.INITIAL_CONTEXT_FACTORY, ldapCF);
         env.put(Context.PROVIDER_URL, ldapURL + ldapBaseDN);
         env.put(Context.SECURITY_PRINCIPAL, ldapUserID);
         env.put(Context.SECURITY_CREDENTIALS, ldapPassword);

      try {
         // Create initial context
         DirContext ctx = new InitialDirContext(env);

         String[] attrIDs = {searchAttr};
         SearchControls ctls = new SearchControls();
         ctls.setReturningAttributes(attrIDs);
```

```
// Specify the search filter
String filter = "(|(cn="+searchCN+")(uid="+searchCN+"))";

// Search for objects using the above filter
NamingEnumeration answer = ctx.search("", filter, ctls);

findCN(answer, doc);

                // Close the context when we're done
                ctx.close();

            } catch(NamingException e) {
                e.printStackTrace();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
} // end of NotesMain

public static void findCN(NamingEnumeration enum, Document doc) {

    try {
        if (enum.hasMore()) {
            while (enum.hasMore()) {
                SearchResult sr = (SearchResult)enum.next();
                System.out.println(">>>" + sr.getName());
                findAttrs(sr.getAttributes(), doc);
            }
} else {
        Item item = doc.getFirstItem("HiddenSearchResults");
        doc.replaceItemValue("HiddenSearchResults", "CN not found");
        doc.save(true);
}
    } catch (NamingException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
} // end of findCN

public static void findAttrs(Attributes attrs, Document doc) {

    try {
        Item item = doc.getFirstItem("HiddenSearchResults");
        Vector v = new Vector();
        String result;

        if (attrs.size() == 0) {
            v.addElement("Name found but attribute is blank or not found. Try again ...");
        } else {
        /* Get each attribute */
            try {
                for (NamingEnumeration ae = attrs.getAll(); ae.hasMore();) {
                    Attribute attr = (Attribute)ae.next();
                    /* Get each value */
                    for (NamingEnumeration e = attr.getAll(); e.hasMore();) {
                        result = (String)e.next();
                        v.addElement(result);
                    }
                }
            } catch (NamingException e) {
                e.printStackTrace();
            }
    } // end if
        doc.replaceItemValue("HiddenSearchResults", v);
        doc.save(true);
    } catch (Exception e) {
        e.printStackTrace();
}
    }
} // end of findAttrs
```

The Search LDAP button code

The following is the complete code example for the Search LDAP button.

```
Sub Click(Source As Button)
    Dim s As New NotesSession
    Dim ws As New NotesUIWorkspace
    Dim db As NotesDatabase
    Dim agent As NotesAgent
    Dim doc As NotesDocument
    Dim uidoc As NotesUIDocument
    Dim searchResultItem As NotesItem
    Dim paramid As String

    Set db = s.CurrentDatabase
    Set uidoc = ws.CurrentDocument
    Set doc = uidoc.Document
    Set agent = db.GetAgent("LDAPSearchWithFilter")

    Call doc.save(True, False)
    paramid = doc.NoteID

    Call agent.RunOnServer(paramid)

    Delete doc

    Set doc = db.GetDocumentByID(paramid) ‘ retrieve the back-end document

    'Get the updated field
    Set searchResultItem = doc.getFirstItem("HiddenSearchResults")
    Forall values In searchResultItem.Values
        Call uidoc.FieldAppendText("SearchResults",  values)
        ‘ Add new line, for results that have multi values
        Call uidoc.FieldAppendText("SearchResults", Chr(10))
    End Forall
    Call uidoc.Refresh

    doc.Remove(True)   ‘you do not want to keep the back-end doc
End Sub
```

The Entering event code for the SearchCN field

The following is the code example for the Entering event of the SearchCN field.

```
    Dim ws As New NotesUIWorkspace
    Dim uidoc As NotesUIDocument
    Set uidoc = ws.CurrentDocument

    Call uidoc.FieldSetText("SearchCN", "")
    Call uidoc.FieldSetText("SearchAttr", "")
    Call uidoc.FieldSetText("SearchResults", "")
    Call uidoc.FieldSetText("SearchResultsGroup", "")
    Call uidoc.FieldSetText("HiddenSearchResults", "")
    If uidoc.FieldGetText("SearchType") = "Group" Then
        Call uidoc.FieldSetText("SearchAttr", "member")
    End If
    Call uidoc.Refresh
```

The Exiting event code for the SearchCN field

The following is the code example for the Exiting event of the SearchCN field. The code is similar to the code for the Search LDAP button, but with additional string processing as follows:

```
'Get the updated field
    Set searchResultItem = doc.getFirstItem("HiddenSearchResults")
    Forall values In searchResultItem.Values
        pos1 = Instr(Cstr(values), "=")
        pos2 = Instr(Cstr(values), ",")
        stringValue =Mid(Cstr(values), pos1+1, pos2-pos1)
        Call uidoc.FieldAppendText("HiddenSearchResults", stringValue)
    End Forall
```