

# How to Manage and Use Dynamic Groups in Novell eDirectory

## How-To Article

NOVELL APPNOTES

**Haripriya S.**  
Software Consultant  
Novell, Inc.  
[sharipriya@novell.com](mailto:sharipriya@novell.com)

**Jaimon Jose**  
Software Consultant  
Novell, Inc.  
[jjaimon@novell.com](mailto:jjaimon@novell.com)

**Jim Sermersheim**  
Sr. Software Engineer  
Novell, Inc.  
[jimse@novell.com](mailto:jimse@novell.com)

This AppNote discusses ways to manage and use dynamic groups in Novell eDirectory to do everyday tasks. It also discusses performance implications and other factors that must be kept in mind in order to put the dynamic groups feature to optimum use.

### Contents:

- Introduction to Dynamic Groups
- Features of Dynamic Groups
- Managing and Using Dynamic Groups Through LDAP
- Example LDAP Commands and LDIF Data for Dynamic Groups Operations
- Performance Considerations
- Other Things to Remember
- Conclusion

Topics	Novell eDirectory, dynamic groups, user management
Products	Novell eDirectory v8.6.1
Audience	network administrators, developers
Level	intermediate
Prerequisite Skills	familiarity with eDirectory, LDAP, LDIF
Operating System	NetWare 6, Solaris, Linux, Windows 2000, Windows NT
Tools	LDAP SDK Utilities
Sample Code	yes

## Introduction to Dynamic Groups

In Novell eDirectory, Group objects are used to define a set of other Directory objects; the objects listed in the group are called the members of the group. You may have created groups of members for the purposes of single-point rights allocation, mail distribution lists, role-based policies, and so on. Before Novell eDirectory v8.6.1, the only way to build the membership list of a group was to explicitly add each member.

Consider an organization that uses group membership lists to control access to its facilities. If a group holding all employees in the Marketing department is used to grant access to a particular facility, and an employee moves from Marketing to Sales, there is extra maintenance involved in updating the Directory. In addition to changing the department attribute of the employee, the administrator has to remove the employee from the group and add him/her as a member of another group. The more groups an employee is part of, the higher the maintenance cost becomes when changes happen.

To ease the administrative task of group maintenance, Novell eDirectory v8.6.1 has introduced the concept of dynamic groups. Dynamic groups let you specify the members of a group using a search filter. The members of a dynamic group are computed dynamically by the eDirectory server(s) whenever the groups are accessed or evaluated. This makes it easier for a user to group objects together because membership can be based on a certain criterion, without having to manually add each member to the Group object.

For the example above, a dynamic group for Marketing would specify that the members are all employees that have “marketing” in their department attribute. This way, only the department attribute of the employee needs to be changed for the move. So when an employee moves from Marketing to Sales, a simple change to the employee’s department attribute is sufficient to make the change in group membership.

In addition to the search filter criteria, the membership list may be further modified by manually excluding or including specific objects.

This AppNote outlines the features of dynamic groups, along with their advantages and limitations. It then explains exactly how to build a dynamic group and how to specify the search criteria that identifies the membership list.

## Features of Dynamic Groups

You may wonder, “What exactly can I do with a dynamic group?”. This section discusses the functionality you’ll get, followed by some things that are not possible with dynamic groups.

With a dynamic group, you get the following features:

- *Membership List.* Applications such as mailing software can read the dynamic membership list, and can off-load the maintenance of membership lists to the server.
- *Member Inclusion.* As with static groups, you can use the compare operation to check whether an asserted object is in the membership list. This can be used to apply an external policy to an object such as a client application that grants or denies access to external resources based on an object's inclusion in a group.
- *Internal Access Control.* Granting permissions to a dynamic group object grants those permissions to its dynamic members also. Administrators can use this for setting internal access control policies.

Some things you can't do are:

- *List of Groups.* When a member is manually added to a static group, the management tool (ConsoleOne, or NWAdmin) will—without you knowing—update a `groupMembership` attribute on the member object to point back at the static group. As long as the management tool performs this correctly and is exclusively used to populate static group membership lists, the `groupMembership` attribute can be used to list all the static groups you're a member of. Obviously, this attribute is not updated whenever an object happens to match or not match a dynamic group's filter. In order to find the dynamic groups you're a member of, you can search for all dynamic groups that have `member=<myDN>`. Note that the list you get back is subject to access controls.
- *Security Attributes.* Similar to `groupMembership`, management tools populate the `securityEquals` attribute on members added to a static group with the group's DN, and populate the `equivalentToMe` attribute on the group with the member's DN. That's a lot of DN populations! These attributes are used by traditional access control methods including the NetWare file system. Again, these attributes are not populated when a member is evaluated for inclusion in a dynamic group. So, if you have a need for these attributes to be populated, this is a limitation. As mentioned earlier, Bindery and NetWare file system access control use these attributes, and thus do not recognize dynamic members when performing their access control calculations.

Additional controls or extensions may be provided in the future to provide additional functionality or provide more user control of existing functionality.

### **Advantages and Limitations of Dynamic Groups**

So why are dynamic groups worth reading about, let alone deploying? This section lists a few advantages dynamic groups provide over traditional (static) groups, and then lists the disadvantages.

**Advantages.** The advantages of dynamic groups can be summarized as follows:

- *References to Objects.* User objects can be created or deleted without having to touch all the groups that the object is a member of. This is particularly useful in the case of non-local members, where the overhead of external references, back-links, and so on can be avoided. Groups can be created before the objects specified by the membership criterion are created.
- *Smaller DIBs.* Because the dynamic members are not stored, dynamic groups will consume less storage when large member lists are involved.
- *Quicker Synchronization.* During synchronization, only the search filter needs to be transferred rather than all the individual members.
- *Centralized Query.* Often, client applications perform a “canned” query to obtain a list of objects. But what happens if that query ever needs to be changed? Rather than recoding, rebuilding, and distributing new client applications, a dynamic group can be employed. The client applications use the dynamic group as the query; thus it can be changed frequently without having to touch the client software.

**Disadvantages.** The disadvantages of dynamic groups are:

- *Referential Integrity.* The search filter holds the DN of the base of the search. Currently, this DN is not protected by referential integrity. Thus, moving, deleting or renaming this object can stop the dynamic groups using it from working.
- *Performance.* Because a query must take place, dynamic group operations will perform slower than static group operations. Be aware that reading the membership list will cause a full search to occur, whereas comparing a specific DN to the membership list will only compare that object to the filter.
- *Rights to Objects.* The NetWare File system and bindery are not yet aware of dynamic groups, thus cannot take advantage of rights granted to the dynamic members.
- *LDAP-Only Administration.* ConsoleOne is currently not aware of dynamic groups; so dynamic group objects cannot be managed using this utility. This AppNote will give examples of how dynamic groups are created and managed through LDAP later.

## Managing and Using Dynamic Groups Through LDAP

As mentioned above, there is no support for dynamic groups in ConsoleOne. Thus Novell suggests that you use LDAP to manage them. The remainder of this AppNote assumes you are using LDAP operations to create and update dynamic groups, and therefore uses LDAP object classes and attribute names.

The object classes `dynamicGroup` and `dynamicGroupAux` are the structural and auxiliary classes that represent a dynamic group. Thus, a dynamic group can be created in eDirectory by adding an object with `objectClass=dynamicGroup`. Alternately, any existing object (including a static group object) can be upgraded to a dynamic group by adding an `objectClass` attribute with the value `dynamicGroupAux`.

**Note:** Keep in mind that auxiliary classes are only supported in eDirectory 8.5 release 85.23 and later, and eDirectory 8.0 release 8.78 and later, and that Dynamic Groups are only supported beginning with eDirectory 8.6.1.

### Attributes Used by Dynamic Groups

The search criterion for the members is specified by the attribute `memberQueryURL`. Other attributes that are used by the dynamic group feature are `uniqueMember`, `excludedMember`, `dgIdentity`, `dgAllowUnknown`, `dgTimeout`, and `dgAllowDuplicates`. Administrators and developers can use the dynamic group features as follows.

The *dynamic group* feature allows you to specify a set of rules as an LDAP URL that will be used to evaluate the group members. The LDAP URL is held in a new attribute called `memberQueryURL` and is accessed by LDAP.

For example, to group together all employees in the Sales department for a mailing list, you would define a dynamic group with the following value for `memberQueryURL`:

```
ldap:///ou=sales,o=myorg??sub?(objectclass=inetOrgPerson)
```

**memberQueryURL.** This holds the entire search query that is used to define the dynamic membership list in the form of an LDAP URL. The format of LDAP URL is specified in RFC 2255 (see <http://ietf.org/rfc/rfc2255.txt>) which consists of the following rough format:

```
ldap://hostname:port/<baseDN>?<attrlist>?<scope>?<search filter>[?<extensions>]
```

The hostname, port, and attrlist have no effect on dynamic groups, and hence the general format of the `memberQueryURL` attribute is as follows:

```
ldap:///<base dn>??<scope>?<search filter>[?x-chain]
```

<base dn>	The distinguished name of the search base. If the base is not specified the root of the tree is assumed.
<scope>	<p>The scope of the search, which can be one of these values:</p> <ul style="list-style-type: none"> <li>• <b>base</b> only searches the base object (&lt;base dn&gt;)</li> <li>• <b>one</b> searches the direct subordinates of the base object. The base object itself is not searched</li> <li>• <b>sub</b> searches the base object and all objects in the subtree below it</li> </ul> <p>If the scope is not specified, the base scope is assumed.</p>

<filter>	An LDAP search filter to apply to entries within the specified scope of the search. Only entries matching the search filter will be selected as the result. If the filter is not specified a filter of "objectclass=*" is assumed. LDAP search filters are described in RFC 2254 RFC 2254 (see <a href="http://ietf.org/rfc/rfc2254.txt">http://ietf.org/rfc/rfc2254.txt</a> ), and most any LDAP reference manual.
<extensions> (x-chain)	A special extension, <i>x-chain</i> is defined for dynamic groups. The <i>x-chain</i> option is used to indicate whether the search for dynamic members should chain across multiple servers, or should be done only on the server containing the dynamic group object. If <i>x-chain</i> is set, then the server will, if needed, communicate with other servers while searching for dynamic members. If the <i>x-chain</i> extension is absent, then the search for dynamic members will not chain across other servers, and return only local results. The use of this extension should be carefully considered, as it can result in lengthy operations. When designing tree structure, you may want to consider ensuring that all possible matches for a dynamic group reside on the local server.  The use of <i>x-chain</i> is optional and the default is to not chain. Currently, the only extension supported in a memberQueryURL is the <i>x-chain</i> extension.

If the memberQueryURL is not present on the dynamic group object, the group will not have any dynamic members. For example, this LDAP URL will result in a list containing all users (inetOrgPerson objects) in the o=acme subtree:

```
memberQueryURL: ldap:///o=acme??sub?objectclass=inetorgperson?x-chain
```

In the event that the subtree is not all held on the local server, the search will communicate with other servers to continue the search.

**Note:** The memberQueryURL attribute can have multiple values, but in eDirectory 8.6.1, only the first value will be used for computing the dynamic members. To avoid any confusion administrators should make sure that only one value is present on this attribute.

**uniqueMember.** A dynamic group is a subtype of a static group. Therefore, in addition to specifying the dynamic members through the memberQueryURL attribute, some members can also be specified statically, using the uniqueMember attribute. This is particularly useful when you want to add some members who do not fall under the criteria for membership.

While listing a dynamic group, the uniqueMember attribute lists both the statically set as well as the dynamic members of the group. This way, existing applications that read group memberships will continue to function when reading members of a dynamic group.

The syntax of uniqueMember is Distinguished Name and can take any DN value. Here is an example:

```
uniqueMember: cn=admin,o=org
```

**Note:** Both uniqueMember and member are mapped to the NDAP member attribute, so either attribute will work here.

**excludedMember.** Just as some members can be added statically to a dynamic group, members can also be excluded statically from a dynamic group. This is useful when you want to deny a few specific objects from being members of a dynamic group, though they may fall under the criteria for membership.

The syntax of `excludedMember` is Distinguished Name and can take any DN value. Here is an example:

```
excludedMember: cn=trainee1,o=org
```

A DN specified in `uniqueMember` will override the same DN specified in `excludedMember`, and as such, that DN will be considered a member of the dynamic group. An entry will be a member of a dynamic group if (a) it is either a static member specified by `uniqueMember`, or (b) it is specified by the search filter and not specified in `excludedMember`.

**dgIdentity.** In order to perform the search specified by the `memberQueryURL`, the server uses a specific identity so that the results will always be consistent. The identity object must have authentication credentials so the server can authenticate as the identity object. The server will decide the identity to be used for a dynamic group (DG) object as follows:

- If there is a `dgIdentity` attribute set on the dynamic group object, then the object specified by the `dgIdentity` attribute will be used as the identity for the dynamic member search.
- If the `dgIdentity` attribute is not present, but DG object has a password attribute, then the **DG object** itself will be used as the identity for dynamic member search.
- If neither the `dgIdentity` attribute nor the password is present, then the search for dynamic members will be done as [Public] (the anonymous user).

The syntax of `dgIdentity` is Distinguished Name and it can be set to any entry in the tree. Here is an example:

```
dgIdentity: cn=localadmin,ou=eng,o=org
```

In order for the dynamic member evaluation to work, the object specified by the `dgIdentity` attribute must be present on the same partition as the dynamic group object. The `dgIdentity` attribute has the schema flags `DS_WRITE_MANAGED` set. This means that one can only specify an object as the `dgIdentity` of a dynamic group, if he has administrative rights on that object. This ensures that users cannot flout the ACL checks and view more objects than creating a dynamic group and setting some other entry as the `dgIdentity` will allow them to.

**dgAllowDuplicates.** While listing the members of a dynamic group, this specifies whether or not duplicates will be found in the member list. Duplicates may occur if an object is found in the search result of the `memberQueryURL`, as well as the `uniqueMember` attribute; but if `dgAllowDuplicates` is not true, then the server will eliminate the duplicates. By allowing duplicates, the administrator can reduce the load on the server while listing dynamic group members.

This is a boolean attribute and can be set to true or false (the default is false). Here is an example:

```
dgAllowDuplicates: true
```

**dgAllowUnknown.** This boolean attribute determines the behavior when the dynamic group members are not fully expandable due to some reason. It determines the inclusion or exclusion of members in the dynamic group when the membership cannot be correctly ascertained.

For example, if the search specified by `memberQueryURL` is not fully done because one of the replicas is not accessible, then if `dgAllowUnknown` is set to true, the object in question will be considered to be a member of the dynamic group. This is also true for dynamic ACL rights computations, and therefore must be used carefully. In short, unless the implications of setting this to true are fully understood, the administrator should always leave it unset or set it to false. When the attribute is not set, it is assumed to be false. Here is an example:

```
dgAllowUnknown: false
```

**dgTimeout.** This integer attribute determines how long to wait to get results from another server during a dynamic groups member search when the search operation spans across servers. The time interval is specified in seconds, and once reached, will terminate the search. Any members found before the search is terminated are included in the list. Here is an example:

```
dgTimeout: 40
```

The behavior of `dgAllowUnknown` is considered when the membership cannot be determined because of a timeout.

## Using Dynamic Groups to Control Access to Resources

You can grant access control permissions to a dynamic group, and have those permissions be applied to all the members of that group.

Earlier, we mentioned that Novell management tools do some things behind the scenes when you add a member to a static group. One of these is to populate the `securityEquals` attribute on the member object to the DN of the group. This allows you to set an ACL on a resource with the trustee (subject) as the static group. If you think about it, with static groups, it's not being part of the group that gives you the rights of the group, it's being security equivalent to the group.

Because dynamic members aren't security equivalent to a dynamic group, you need to add a special flag to any ACL that points to a dynamic group. This way, when a user tries to perform some privileged operation on the resource object, eDirectory will know that it must check to see if the user is a member of the dynamic group as opposed to checking the user's security equivalences.

The next section describes how to set this special "dynamic ACL."

### Dynamic ACL

When using LDAP to create an ACL that names a dynamic group as the trustee (subject), the privilege bit `0x20000000` must be set by bit-wise ORing (or adding) the value with the other privileges being set.

As an example, say you have an object named `cn=prn1,o=org` and want to grant read and compare privileges on all attributes for the trustee `cn=dgl,o=org`:

- Privileges required = Read and Compare = 2 + 1 = 3
- Dynamic ACL value = `0x20000000` = 536870912
- Privileges to be given for a dynamic ACL for Read/Compare = 536870912 + 3 = 536870915

So the ACL value to be added for the dynamic ACL on the object `cn=prn1,o=org` will be:

ACL: `536870915#entry#cn=dgl,o=org#[All Attributes Rights]`

This dynamic ACL bit is also defined in Novell's LDAP C SDK and can be used for the bit-wise OR operation. The following table shows which privilege bits are used to specify each privilege for a dynamic ACL.

Rights On	Required Rights	Normal ACL Permissions	Dynamic ACL Permissions
Entry rights	Browse	1 (0x1)	536870913 (0x20000001)
	Add	2 (0x2)	536870914 (0x20000002)
	Delete	4 (0x4)	536870916 (0x20000004)
	Rename	8 (0x8)	536870920 (0x20000008)
	Supervisor	16 (0x10)	536870928 (0x20000010)

Rights On	Required Rights	Normal ACL Permissions	Dynamic ACL Permissions
Attributes Rights	Compare	1 (0x1)	536870913 (0x20000001)
	Read	2 (0x2)	536870914 (0x20000002)
	Write	4 (0x4)	536870916 (0x20000004)
	Add Self	8 (0x8)	536870920 (0x20000008)
	Supervisor	32 (0x20)	536870944 (0x20000020)

If you assign a dynamic group as a trustee, and don't add in the dynamic bit, only the static members of the dynamic group (as specified using the securityEquals relation) will be able to inherit those privileges.

For example, if `cn=dg1,o=myorg` is a dynamic group, `cn=bob,o=myorg` is a static member of that group, and `cn=alice,o=myorg`, and `cn=sandy,o=myorg` are dynamic members of the group, the following ACL will grant write rights to all attributes only for `cn=bob,o=myorg`:

```
ACL: 4#entry#cn=dg1,o=myorg#[All Attributes Rights]
```

But the next ACL will grant all the three users (bob, alice and sandy), write access to all attributes:

```
ACL: 536870916#entry#cn=dg1,o=myorg#[All Attributes Rights]
```

For more information on eDirectory LDAP ACLs, see <http://ietf.org/internet-drafts/draft-sermersheim-nds-ldap-schema-02.txt>.

## Errors and Result Codes

In the course of expanding dynamic members for listing a dynamic group, the server might encounter an error. In that case, if the server is capable of finding some but not all the dynamic members, an Unknown Error is returned by LDAP, with additional information displaying the NDAP error "error partial results (-6016)". The most common reason for a search or read operation on a dynamic group returning partial results is that *x-chain* is set, but one of the search references encountered during the dynamic member search is not accessible. This could have happened because the server holding the relevant replica is down. In this case the server will try to list all the members that it could evaluate, and then return an error. The client applications can treat this error as a non-fatal error and list the partial results of the operation.

## Example LDAP Commands and LDIF Data for Dynamic Groups Operations

The examples in this section use LDAP utilities like *ldapmodify* (or ICE) and *ldapsearch.LDIF* (LDAP data interchange format) provides a simple way to manage and use dynamic groups functionality. These LDAP tools are available in the LDAP SDK from Novell. The ICE utility is installed with ConsoleOne and is available as a command-line utility and as a wizard. For usage instructions, refer to the help that comes with these utilities.

- LDIF to add a dynamic group

```
dn: cn=dg1,o=myorg
changetype: add
objectClass: dynamicGroup
memberQueryURL: ldap:///o=myorg??sub?cn=*
```

- LDIF to change a group object to a dynamic group object (with x-chain set)

```
dn: cn=group,o=myorg
changetype: modify
add: objectClass
objectClass: dynamicGroupAux
-
add: memberQueryURL
memberQueryURL: ldap:///o=myorg??sub?cn=?*x-chain
```

The above example also specifies that chaining to other servers should be done if a non-local reference is encountered during the search.

- LDAP search utility example of listing the members of a dynamic group

```
ldapsearch -b o=myorg -s one "cn=dg1" uniquemember
```

Results:

```
dn: cn=dg1,o=myorg
uniquemember: cn=admin,o=myorg
uniquemember: cn=bob,ou=finance,o=myorg
uniquemember: cn=alice,ou=finance,o=myorg
```

- LDAP command for listing the static and dynamic groups under o=myorg of which cn=bob,ou=finance,o=myorg is a member

```
ldapsearch -b o=myorg -s sub"member=cn=bob,ou=finance,o=myorg" dn
```

Results:

```
dn: cn=dg1,o=myorg
dn: cn=sg1,o=myorg
dn: cn=dg2,ou=finance,o=myorg
```

- LDAP command for listing all static and dynamic groups under o=myorg which have at least one member

```
ldapsearch -b o=myorg -s sub "member=*" dn
```

Results:

```

dn: cn=dg1,o=myorg
dn: cn=sg1,o=myorg
dn: cn=dg2,ou=finance,o=myorg
dn: cn=dg2,o=myorg
dn: cn=dg3,ou=finance,o=myorg
dn: cn=sg2,ou=finance,o=myorg

```

- LDAP command for comparing an entry against a dynamic group to assert membership

```

ldapsearch -b cn=dg1,o=myorg -s base
           "member=cn=bob,ou=finance,o=myorg" dn

```

- LDIF to add a dynamic group using itself as the identity

```

dn: cn=dg2,ou=finance,o=myorg
changetype: add
objectclass: dynamicgroup
memberqueryurl: ldap:///o=myorg??sub?cn=*
userpassword: secret

```

- LDIF to add a dynamic group using cn=admin,o=myorg as the identity

```

dn: cn=dg2,o=myorg
changetype: add
objectclass: dynamicgroup
memberqueryurl: ldap:///o=myorg??sub?cn=*
dgidentity: cn=admin,o=myorg

```

- LDIF to add a dynamic group which excludes *alice* from being a member

```

dn: cn=dg3,ou=finance,o=myorg
changetype: add
objectclass: dynamicgroup
memberqueryurl: ldap:///o=myorg??sub?cn=*
excludedmember: cn=alice,ou=finance,o=myorg

```

- LDIF to add a dynamic ACL to a printer object giving entry rename permissions to dg3

```

dn: cn=prn1,o=myorg
changetype: modify
add: ACL
ACL: 536870920#entry#cn=dg3,ou=finance,o=myorg#[Entry Rights]

```

Now, an attempt to modify the RDN of the object cn=prn1,o=myorg as cn=bob,ou=finance,o=myorg will succeed, whereas the same operation performed as cn=alice,ou=finance,o=myorg will fail because of insufficient access permissions.

## Performance Considerations

As you have seen, dynamic groups can be very powerful. But with that power comes the need for some awareness and responsibility. Remember that any time the membership list is expanded, you're causing eDirectory to perform a search operation. This may not seem costly at first, but what if hundreds of users are continuously reading the member list of a dynamic group?

Here are a few specific examples of potential performance problems and what you can do to prevent them:

- *Listing All Groups I'm a Member Of.* Performing a tree-wide search for “member=<cn=me,o=acme>” may be expensive if there are many dynamic groups. The search itself isn't too bad because eDirectory just asserts that the member object in question matches the memberQueryURL (it doesn't fully expand the membership list); but with too many dynamic groups present in the search scope, this will cause a membership assertion to be performed for each of these groups. Also, there is a problem with what is returned. If all attributes are asked for, then the entire membership list will be expanded for the matching group entries, which may lead to an expensive operation. Be sure to only ask for the attributes you need to see. In this case, you are only interested in the name of the groups themselves, so specify that you want no attributes to be returned. In LDAP this is done by using the attribute “1.1” or “dn” in the attributes list.
- *Listing the Members of a Dynamic Group.* Obviously this is as expensive as performing a search, using the filter held in the memberQueryURL of the dynamic group. If you are obtaining the list for comparison purposes, save yourself and the server some headaches and simply use the compare operation.
- *Dynamic ACLs.* These can become computation intensive and can take some time, particularly if they are located as *subtree* ACLs at a top-level container. As much as possible, use dynamic ACLs as “entry” level ACLs and not as “subtree” ACLs. If a *subtree* ACL must be used, make sure it is set on the lowest level container required. Also, it's best to employ a policy that requires that the subtree specified by the base dn and scope of the memberQueryURL is all held in a single partition, or is ensured to be all held on any server holding the dynamic group. This way, there is no need to spend time communicating with other servers and dealing with network latencies when simply trying to evaluate an ACL.

Remember, dynamic groups are a great way to ease your administration jobs, but like any run-time feature, there is a price to pay in terms of performance. Dynamic groups and dynamic ACLs will be slower than normal groups and ACLs, so use them judiciously.

## Other Things to Remember

Here are a few more things to remember when working with dynamic groups:

- Currently, referential Integrity is not supported for distinguished names that are used inside the `memberQueryURL` attribute. So, if a particular DN is used as the base DN or in the search filter in the `memberQueryURL`, and it is later moved or deleted, it will not be reflected in the `memberQueryURL`. Therefore the result set may be different that what you expected.
- Currently, if multiple values are stored in the `memberQueryURL` attribute only the first that is read will be used. So make sure that you do not set multiple values for that attribute.
- File system rights cannot be managed through dynamic ACLs. Bindery operations do not use rights due to dynamic ACLs.
- The following operations do not use dynamic ACL rights because they would imply recursively evaluating dynamic groups:
  - You cannot depend on the `dgIdentity` object to have rights to do the dynamic member search by getting rights through a dynamic ACL. It must have a normal ACL which grants it rights to perform the search. In addition to browse rights, this also means that the object must have attribute compare rights on all the attributes specified in the `memberQueryURL` search filter.
  - If a filter of `"member=<someDN>"` or `"member=*"` is used in a `memberQueryURL`, it will match only groups having the given DN's as static members.
  - If another group (dynamic or static) is a member of a dynamic group, its members are not added to the membership list.
- `memberQueryURL` is an LDAP attribute name. Using NDAP, the attribute is called `memberQuery`, and it is of a different format. It is advisable to change and view the `memberQueryURL` attribute through LDAP rather than use the `memberQuery` attribute via NCPs.
- Currently, ConsoleOne does not provide snap-ins for managing dynamic groups. This means that you are not able to create or edit the `memberQueryURL` for dynamic group objects. However, you will be able to see the dynamic member list of a dynamic group using ConsoleOne by examining the `member` attribute of the object.

## Conclusion

Hopefully, you now have a working understanding of what dynamic groups are, and how to use and manage them. While dynamic groups provide an intuitive way to manage groups of objects, and ease administrative tasks for large directories, this AppNote has probably only scratched the surface of the many real-world uses you'll come up with on your own.

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.