Search Filter Syntax

В этой статье

Search filters enable you to define search criteria and provide more efficient and effective searches.

ADSI supports the LDAP search filters as defined in RFC2254. These search filters are represented by Unicode strings. The following table lists some examples of LDAP search filters.

Search filter	Description
"(objectClass=*)"	All objects.
"(&(objectCategory=person)(objectClass=user)(!(cn=andy)))"	All user objects but "andy".
"(sn=sm*)"	All objects with a surname that starts with "sm".
"(&(objectCategory=person)(objectClass=contact)(|(sn=Smith)(sn=Johnson)))"	All contacts with a surname equal to "Smith" or "Johnson".

These search filters use one of the following formats.

```
C++

<filter>=(<attribute><operator><value>)
```

or

```
C++

(<operator><filter1><filter2>)
```

The ADSI search filters are used in two ways. They form a part of the LDAP dialect for submitting queries through the OLE DB provider. They are also used with the IDirectorySearch interface.

Operators

The following table lists frequently used search filter operators.

Logical operator	Description
=	Equal to
~=	Approximately equal to
<=	Lexicographically less than or equal to
>=	Lexicographically greater than or equal to
&	AND
|	OR
!	NOT

In addition to the operators above, LDAP defines two matching rule object identifiers (OIDs) that can be used to perform bitwise comparisons of numeric values. Matching rules have the following syntax.

```C++
<attribute name>:<matching rule OID>:=<value>
```

"" is the **lDAPDisplayName** of the attribute, "" is the OID for the matching rule, and "" is the value to use for comparison. Be aware that spaces cannot be used in this string. "" must be a decimal number; it cannot be a hexadecimal number or a constant name such as **ADS_GROUP_TYPE_SECURITY_ENABLED**. For more information about the available Active Directory attributes, see All Attributes.

The following table lists the matching rule OIDs implemented by LDAP.

Matching rule OID	String identifier (from Ntldap.h)	Description
1.2.840.113556.1.4.803	**LDAP_MATCHING_RULE_BIT_AND**	A match is found only if all bits from the attribute match the value. This rule is equivalent to a bitwise **AND** operator.
1.2.840.113556.1.4.804	**LDAP_MATCHING_RULE_BIT_OR**	A match is found if any bits from the attribute match the value. This rule is equivalent to a bitwise **OR** operator.

Matching rule OID	String identifier (from Ntldap.h)	Description
1.2.840.113556.1.4.1941	**LDAP_MATCHING_RULE_IN_CHAIN**	This rule is limited to filters that apply to the DN. This is a special "extended" match operator that walks the chain of ancestry in objects all the way to the root until it finds a match.

The following example query string searches for group objects that have the **ADS_GROUP_TYPE_SECURITY_ENABLED** flag set. Be aware that the decimal value of **ADS_GROUP_TYPE_SECURITY_ENABLED** (0x80000000 = 2147483648) is used for the comparison value.

```C++
(&(objectCategory=group)(groupType:1.2.840.113556.1.4.803:=2147483648))
```

The **LDAP_MATCHING_RULE_IN_CHAIN** is a matching rule OID that is designed to provide a method to look up the ancestry of an object. Many applications using AD and AD LDS usually work with hierarchical data, which is ordered by parent-child relationships. Previously, applications performed transitive group expansion to figure out group membership, which used too much network bandwidth; applications needed to make multiple roundtrips to figure out if an object fell "in the chain" if a link is traversed through to the end.

An example of such a query is one designed to check if a user "user1" is a member of group "group1". You would set the base to the user DN `(cn=user1, cn=users, dc=x)` and the scope to `base`, and use the following query.

```C++
(memberof:1.2.840.113556.1.4.1941:=cn=Group1,OU=groupsOU,DC=x)
```

Similarly, to find all the groups that "user1" is a member of, set the base to the groups container DN; for example `(OU=groupsOU, dc=x)` and the scope to `subtree`, and use the following filter.

```C++
(member:1.2.840.113556.1.4.1941:=cn=user1,cn=users,DC=x)
```

Note that when using **LDAP_MATCHING_RULE_IN_CHAIN**, scope is not limited—it can be `base`, `one-level`, or `subtree`. Some such queries on subtrees may be more processor intensive, such as chasing links with a high fan-out; that is, listing all the groups that a user is a member of. Inefficient searches will log appropriate event log messages, as with any other type of query.

Wildcards

You can also add wildcards and conditions to an LDAP search filter. The following examples show substrings that can be used to search the directory.

Get all entries:

C++

```
(objectClass=*)
```

Get entries containing "bob" somewhere in the common name:

C++

```
(cn=*bob*)
```

Get entries with a common name greater than or equal to "bob":

C++

```
(cn>='bob')
```

Get all users with an email attribute:

C++

```
(&(objectClass=user)(email=*))
```

Get all user entries with an email attribute and a surname equal to "smith":

C++

```
(&(sn=smith)(objectClass=user)(email=*))
```

Get all user entries with a common name that starts with "andy", "steve", or "margaret":

C++

```
(&(objectClass=user)(| (cn=andy*)(cn=steve*)(cn=margaret*)))
```

Get all entries without an email attribute:

C++

```
(!(email=*))
```

The formal definition of the search filter is as follows (from RFC 1960):

```C++
<filter> ::= '(' <filtercomp> ')'
<filtercomp> ::= <and> | <or> | <not> | <item><and> ::= '&' <filterlist>
<or> ::= '|' <filterlist>
<not> ::= '!' <filter>
<filterlist> ::= <filter> | <filter> <filterlist>
<item>::= <simple> | <present> | <substring>
<simple> ::= <attr> <filtertype> <value><filtertype> ::= <equal> | <approx> | <ge> | <le>
<equal> ::= '='
<approx> ::= '~='
<ge> ::= '>='
<le> ::= '<='
<present> ::= <attr> '=*'
<substring> ::= <attr> '=' <initial> <any> <final>
<initial> ::= NULL | <value><any> ::= '*' <starval>
<starval> ::= NULL | <value>'*' <starval>
<final> ::= NULL | <value>
```

The token is a string that represents an AttributeType. The token is a string that represents an AttributeValue whose format is defined by the underlying directory service.

If a must contain the asterisk (*), left parenthesis ((), or right parenthesis ()) character, the character should be preceded by the backslash escape character ().

Special Characters

If any of the following special characters must appear in the search filter as literals, they must be replaced by the listed escape sequence.

ASCII character	Escape sequence substitute
*	\2a
(	\28
)	\29
\	\5c
NUL	\00
/	\2f

Note

In cases where a MultiByte Character Set is being used, the escape sequences listed above must be used if the search is performed by ADO with the SQL dialect.

In addition, arbitrary binary data may be represented by using the escape sequence syntax by encoding each byte of binary data with the backslash () followed by two hexadecimal digits. For example, the four-byte value 0x00000004 is encoded as \00\00\00\04 in a filter string.