Platform SDK: Active Directory

How to Specify Comparison Values

Each attribute type has a syntax that determines the type of comparison values that you can specify in a search filter for that attribute.

The following sections describe requirements for each attribute syntax. For more information about attribute syntaxes, see Syntaxes for Active Directory Attributes.

Boolean

The value specified in a filter must be a string value that is either `TRUE` or `FALSE`. The following examples show how to specify a Boolean comparison string.

The following example will search for objects that have a `showInAdvancedViewOnly` set to TRUE:

```
(showInAdvancedViewOnly=TRUE)
```

The following example will search for objects that have a `showInAdvancedViewOnly` set to FALSE:

```
(showInAdvancedViewOnly=FALSE)
```

Integer and Enumeration

The value specified in a filter must be a decimal Integer. Hexadecimal values must be converted to decimal. A value comparison string takes the following form.

```
<attribute name>:<value>
```

`<attribute name>` is the `lDAPDisplayName` of the attribute and `<value>` is the value to use for comparison.

The following code example shows a filter that will search for objects that have a `groupType` value that is equal to the ADS_GROUP_TYPE_UNIVERSAL_GROUP (8) flag and the ADS_GROUP_TYPE_SECURITY_ENABLED (0x80000000) flag. The two flags combined equal 0x80000008, which converted to decimal is 2147483650.

```
(groupType=2147483650)
```

The LDAP matching rule operators can also be used to perform bitwise comparisons. For more information about matching rules, see Search Filter Syntax. The following code example shows a filter that will search for objects that have a `groupType` with the ADS_GROUP_TYPE_SECURITY_ENABLED (0x80000000 = 2147483648) bit set.

```
(groupType:1.2.840.113556.1.4.803:=2147483648))
```

OctetString

The value specified in a filter is the data to be found. The data must be represented as a two character encoded byte string where each byte is preceded by a backslash (\). For example, the value 0x05 will appear in the string as `\05`.

The ADsEncodeBinaryData function can be used to create an encoded string representation of binary data. The `ADsEncodeBinaryData` function does not encode byte values that represent alpha-numeric characters. It will, instead, place the character into the string without encoding it. This results in the string containing a mixture of encoded and unencoded characters. For example, if the binary data is 0x05|0x1A|0x1B|0x43|0x32, the encoded string will contain `\05\1A\1BC2`. This has no effect on the filter and the search filters will work correctly with these types of strings.

Wildcards are accepted.

The following code example shows a filter that contains encoded string for `schemaIDGUID` with GUID value of `{BF967ABA-0DE6-11D0-A285-00AA003049E2}`:

```
(schemaidguid=\BA\7A\96\BF\E6\0D\D0\11\A2\85\00\AA\00\30\49\E2)
```

Sid

The value specified in a filter is the encoded byte string representation of the SID. For more information about encoded byte strings, see the previous section in this topic which discusses OctetString syntax.

The following code example shows a filter that contains an encoded string for `objectSid` with SID string value of `S-1-5-21-1935655697-308236825-1417001333`:

```
(ObjectSid=\01\04\00\00\00\00\00\05\15\00\00\00\11\C3\5Fs\19R\5F\12u\B9uT)
```

DN

The entire distinguished name, to be matched, must be supplied.

Wildcards are not accepted.

Be aware that the objectCategory attribute also enables you to specify the lDAPDisplayName of the class set on the attribute.

The following example shows a filter that specifies a member that contains `CN=TestUser,DC=Fabrikam,DC=COM`**:**

```
(member=CN=TestUser,DC=Fabrikam,DC=COM)
```

INTEGER8

The value specified in a filter must be a decimal integer. Convert hexadecimal values to decimal.

The following code example shows a filter that specifies a creationTime set to a FILETIME of `3/10/99 3:31:32 PM`**:**

```
(creationTime=125655822921406250)
```

The following functions create an exact match (=) filter for a large integer attribute and verify the attribute in the schema and its syntax:

[C++]

```
//*************************************************************************
//
//  CheckAttribute()
//
//*************************************************************************

HRESULT CheckAttribute(LPOLESTR szAttribute, LPOLESTR szSyntax)
{
    HRESULT hr = E_FAIL;
    BSTR bstr;
    IADsProperty *pObject = NULL;
    LPWSTR szPrefix = L"LDAP://schema/";
    LPWSTR szPath;

    if((!szAttribute) || (!szSyntax))
    {
        return E_POINTER;
    }

    // Allocate a buffer large enough to hold the ADsPath of the attribute.
    szPath = new WCHAR[lstrlenW(szPrefix) + lstrlenW(szAttribute) + 1];
    if(NULL == szPath)
    {
        return E_OUTOFMEMORY;
    }

    // Create the ADsPath of the attribute.
    lstrcpyW(szPath, szPrefix);
    lstrcatW(szPath, szAttribute);

    hr = ADsOpenObject( szPath,
                        NULL,
                        NULL,
                        ADS_SECURE_AUTHENTICATION, // Use Secure Authentication.
                        IID_IADsProperty,
                        (void**)&pObject);
    if(SUCCEEDED(hr))
    {
        hr = pObject->get_Syntax(&bstr);
        if (SUCCEEDED(hr))
        {
            if (0==lstrcmpiW(bstr, szSyntax))
            {
                hr = S_OK;
            }
            else
            {
                hr = S_FALSE;
            }
```

```
        }

        SysFreeString(bstr);
    }

    if(pObject)
    {
        pObject->Release();
    }

    delete szPath;

    return hr;
}

//+--------------------------------------------------------------------------
//
//  CreateExactMatchFilterLargeInteger()
//
//+--------------------------------------------------------------------------

HRESULT CreateExactMatchFilterLargeInteger( LPOLESTR szAttribute,
                                            INT64 liValue,
                                            LPOLESTR *pszFilter)
{
    HRESULT hr = E_FAIL;

    if ((!szAttribute) || (!pszFilter))
    {
        return E_POINTER;
    }

    // Verify that the attribute exists and has
    // Integer8 (Large Integer) syntax.

    hr = CheckAttribute(szAttribute, L"Integer8");
    if (S_OK == hr)
    {
        LPWSTR szFormat = L"%s=%I64d";
        LPWSTR szTempFilter = new WCHAR[lstrlenW(szFormat) + lstrlenW(szAttribute) + 20 + 1];

        if(NULL == szTempFilter)
        {
            return E_OUTOFMEMORY;
        }

        swprintf(szTempFilter, L"%s=%I64d", szAttribute, liValue);

        // Allocate buffer for the filter string.
        // Caller must free the buffer using CoTaskMemFree.
        *pszFilter = (OLECHAR *)CoTaskMemAlloc(sizeof(OLECHAR) * (lstrlenW(szTempFilter) + 1));
        if (*pszFilter)
        {
            wcscpy(*pszFilter, szTempFilter);
            hr = S_OK;
        }
        else
        {
            hr = E_OUTOFMEMORY;
        }

        delete szTempFilter;
    }

    return hr;
}
```

PrintableString

Attributes with these syntaxes should adhere to specific character sets. For more information, see Syntaxes for Active Directory Attributes. Currently, Active Directory does not enforce those character sets.

The value specified in a filter is a string. The comparison is case-sensitive.

GeneralizedTime

The value specified in a filter is a string that represents the date in the following form:

```
YYYYMMDDHHMMSS.0Z
```

`0Z` **indicates no time differential. Be aware that Active Directory stores date/time as Greenwich Mean Time (GMT). If a time differential is not specified, the default is GMT.**

If the local time zone is not GMT, use a differential value to specify your local time zone and apply the differential to GMT. The differential is based on: GMT=Local+differential.

To specify a differential, use:

```
YYYYMMDDHHMMSS.0[+/-]HHMM
```

The following example shows a filter that specifies a `whenCreated` time set to 3/23/99 8:52:58 PM GMT:

```
(whenCreated=19990323205258.0Z)
```

The following example shows a filter that specifies a `whenCreated` time set to 3/23/99 8:52:58 PM New Zealand Standard Time (differential is +12 hours):

```
(whenCreated=19990323205258.0+1200)
```

The following code example shows how to calculate time zone differential. The function returns the differential between the current local time zone and GMT. The value returned is a string in the following format:

```
[+/-]HHMM
```

For example, Pacific Standard Time is -0800.

[C++]

```
//*****************************************************************************
//
//  GetLocalTimeZoneDifferential()
//
//*****************************************************************************

HRESULT GetLocalTimeZoneDifferential(LPOLESTR *pszDifferential)
{
    if(NULL == pszDifferential)
    {
        return E_INVALIDARG;
    }

    HRESULT hr = E_FAIL;
    DWORD dwReturn;
    TIME_ZONE_INFORMATION timezoneinfo;
    LONG lTimeDifferential;
    LONG lHours;
    LONG lMinutes;

    dwReturn  = GetTimeZoneInformation(&timezoneinfo);

    switch (dwReturn)
    {
    case TIME_ZONE_ID_STANDARD:
        lTimeDifferential = timezoneinfo.Bias + timezoneinfo.StandardBias;

        // Bias is in minutes. Calculate the hours for HHMM format.
        lHours =  -(lTimeDifferential/60);

        // Bias is in minutes. Calculate the minutes for HHMM format.
        lMinutes = lTimeDifferential%60L;

        hr = S_OK;
        break;

    case TIME_ZONE_ID_DAYLIGHT:
        lTimeDifferential = timezoneinfo.Bias + timezoneinfo.DaylightBias;
```

```
            // Bias is in minutes. Calculate the hours for HHMM format.
            // Apply the additive inverse.
            // Bias is based on GMT=Local+Bias.
            // A differential, based on GMT=Local -Bias, is required.
            lHours = -(lTimeDifferential/60);

            // Bias is in minutes. Calculate the minutes for HHMM format.
            lMinutes = lTimeDifferential%60L;

            hr = S_OK;
            break;

        case TIME_ZONE_ID_INVALID:
        default:
            hr = E_FAIL;
            break;
        }

        if (SUCCEEDED(hr))
        {
            // The caller must free the memory using CoTaskMemFree.
            *pszDifferential = (OLECHAR *)CoTaskMemAlloc(sizeof(OLECHAR) * (3 + 2 + 1));
            if (*pszDifferential)
            {
                swprintf(*pszDifferential, L"%+03d%02d", lHours, lMinutes);

                hr = S_OK;
            }
            else
            {
                hr = E_OUTOFMEMORY;
            }
        }

        return hr;
    }
```

UTCTime

The value specified in a filter is a string that represents the date in the following form:

```
YYMMDDHHMMSSZ
```

Z indicates no time differential. Be aware that Active Directory stores date and time as GMT time. If a time differential is not specified, GMT is the default.

The seconds value (SS) is optional.

If GMT is not the local time zone, apply a local differential value to specify your local time zone. The differential is: GMT=Local+differential.

To specify a differential, use the following form:

```
YYMMDDHHMMSS[+/-]HHMM
```

The following example shows a filter that specifies a myTimeAttrib time set to 3/23/99 8:52:58 PM GMT:

```
(myTimeAttrib=990323205258Z)
```

The following example shows a filter that specifies a myTimeAttrib time set to 3/23/99 8:52:58 PM without seconds specified:

```
(myTimeAttrib=9903232052Z)
```

The following example shows a filter that specifies a myTimeAttrib time set to 3/23/99 8:52:58 PM New Zealand Standard Time (differential is 12 hours). This is equivalent to 3/23/99 8:52:58 AM GMT.

```
(myTimeAttrib=990323205258+1200)
```

DirectoryString

The value specified in a filter is a string. DirectoryString can contain Unicode characters. The comparison is case-insensitive.

OID

The entire OID to be matched must be supplied.

Wildcards are not accepted.

The objectCategory attribute enables you to specify the lDAPDisplayName of the class set for the attribute.

The following example shows a filter that specifies governsID for volume class:

```
(governsID=1.2.840.113556.1.5.36)
```

Two equivalent filters that specifies systemMustContain attribute containing uNCName, which has an OID of 1.2.840.113556.1.4.137:

```
(SystemMustContain=uNCName)

(SystemMustContain=1.2.840.113556.1.4.137)
```

Other Syntaxes

The following syntaxes are evaluated in a filter similar to an octet string:

- **ObjectSecurityDescriptor**
- **AccessPointDN**
- **PresentationAddresses**
- **ReplicaLink**
- **DNWithString**
- **DNWithOctetString**
- **ORName**

Last updated: January 2005 | What did you think of this topic? | Order a Platform SDK CD