The COM Elevation Moniker - Win32 apps

By stevewhims

Archived: 2026-04-02 10:59:51 UTC

The COM elevation moniker allows applications that are running under user account control (UAC) to activate COM classes with elevated privileges. For more information, see Focus on Least Privilege.

When to Use the Elevation Moniker

The elevation moniker is used to activate a COM class to accomplish a specific and limited function that requires elevated privileges, such as changing the system date and time.

Elevation requires participation from both a COM class and its client. The COM class must be configured to support elevation by annotating its registry entry, as described in the Requirements section. The COM client must request elevation by using the elevation moniker.

The elevation moniker is not intended to provide application compatibility. For example, if you want to run a legacy COM application such as WinWord as an elevated server, you should configure the COM client executable to require elevation, rather than activating the legacy application's class with the elevation moniker. When the elevated COM client calls **CoCreateInstance** using the legacy application's CLSID, the client's elevated state will flow to the server process.

Not all COM functionality is compatible with elevation. The functionality that will not work includes:

- Elevation does not flow from a client to a remote COM server. If a client activates a remote COM server with the elevation moniker, the server will not be elevated, even if it supports elevation.
- If an elevated COM class uses impersonation during a COM call, it might lose its elevated privileges during the impersonation.
- If an elevated COM server registers a class in the running object table (ROT), the class will not be available to non-elevated clients.
- A process elevated by using the UAC mechanism does not load per-user classes during COM activations. For COM applications, this means that the application's COM classes must be installed in the **HKEY_LOCAL_MACHINE** registry hive if the application is to be used both by non-privileged and privileged accounts. The application's COM classes need only be installed in the **HKEY_USERS** hive if the application is never used by privileged accounts.
- Drag and drop is not allowed from non-elevated to elevated applications.

Requirements

In order to use the elevation moniker to activate a COM class, the class must be configured to run as the launching user or the 'Activate as Activator' application identity. If the class is configured to run under any other identity, the

activation returns the error CO_E_RUNAS_VALUE_MUST_BE_AAA.

The class must also be annotated with a "friendly" display name that is multilingual user interface (MUI) compatible. This requires the following registry entry:

```
HKEY_LOCAL_MACHINE\Software\Classes\CLSID
  {CLSID}
     LocalizedString = displayName
```

If this entry is missing, the activation returns the error CO_E_MISSING_DISPLAYNAME. If the MUI file is missing, the error code from the **RegLoadMUIStringW** function is returned.

Optionally, to specify an application icon to be displayed by the UAC user interface, add the following registry key:

```
HKEY_LOCAL_MACHINE\Software\Classes\CLSID
  {CLSID}
     Elevation
        IconReference = applicationIcon
```

IconReference uses the same format as **LocalizedString**:

@pathtobinary,-resourcenumber

In addition, the COM component must be signed for the icon to be displayed.

The COM class must also be annotated as LUA-Enabled. This requires the following registry entry:

```
HKEY_LOCAL_MACHINE\Software\Classes\CLSID
  {CLSID}
     Elevation
        Enabled = 1
```

If this entry is missing, then the activation returns the error CO_E_ELEVATION_DISABLED.

Note that these entries must exist in the HKEY_LOCAL_MACHINE hive, not the HKEY_CURRENT_USER or HKEY_USERS hive. This prevents users from elevating COM classes that they did not also have the privileges to register.

The Elevation Moniker and the Elevation UI

If the client is already elevated, using the elevation moniker will not cause the Elevation UI to display.

How to Use the Elevation Moniker

The elevation moniker is a standard COM moniker, similar to the session, partition, or queue monikers. It directs an activation request to a specified server with the specified elevation level. The CLSID to be activated appears in the moniker string.

The elevation moniker supports the following Run level tokens:

1. Administrator
2. Highest

The syntax for this is as follows:

```
 Elevation:Administrator!new:{guid}
Elevation:Highest!new:{guid}
```

The preceding syntax uses the "new" moniker to return an instance of the COM class specified by *guid*. Note that the "new" moniker internally uses the **IClassFactory** interface to obtain a class object and then calls **IClassFactory::CreateInstance** on it.

The elevation moniker can also be used to get a class object, which implements **IClassFactory**. The caller then calls **CreateInstance** to get an object instance. The syntax for this is as follows:

```
 Elevation:Administrator!clsid:{guid}
```

Sample code

The following code example shows how to use the elevation moniker. It assumes that you have already initialized COM on the current thread.

```
 HRESULT CoCreateInstanceAsAdmin(HWND hwnd, REFCLSID rclsid, REFIID riid, __out void ** ppv)
{
    BIND_OPTS3 bo;
    WCHAR  wszCLSID[50];
    WCHAR  wszMonikerName[300];

    StringFromGUID2(rclsid, wszCLSID, sizeof(wszCLSID)/sizeof(wszCLSID[0]));
    HRESULT hr = StringCchPrintf(wszMonikerName, sizeof(wszMonikerName)/sizeof(wszMonikerName[0]), L"Elevation:A
    if (FAILED(hr))
        return hr;
    memset(&bo, 0, sizeof(bo));
    bo.cbStruct = sizeof(bo);
    bo.hwnd = hwnd;
    bo.dwClassContext  = CLSCTX_LOCAL_SERVER;
    return CoGetObject(wszMonikerName, &bo, riid, ppv);
}
```

BIND_OPTS3 is new in Windows Vista. It is derived from **BIND_OPTS2**.

The only addition is an **HWND** field, **hwnd**. This handle represents a window that becomes the owner of the Elevation UI, if applicable.

If **hwnd** is **NULL**, COM will call GetActiveWindow to find a window handle associated with the current thread. This case might occur if the client is a script, which cannot fill in a **BIND_OPTS3** structure. In this case, COM will try to use the window associated with the script thread.

Over-The-Shoulder (OTS) Elevation

Over-the-shoulder (OTS) elevation refers to the scenario in which a client runs a COM server with an administrator's credentials rather than his or her own. (The term "over the shoulder" means that the administrator is watching over the client's shoulder as the client runs the server.)

This scenario might cause a problem for COM calls into the server, because the server might not call **CoInitializeSecurity** either explicitly (that is, programmatically) or implicitly (that is, declaratively, using the registry). For such servers, COM computes a security descriptor that allows only SELF, SYSTEM, and Builtin\Administrators to makes COM calls into the server. This arrangement will not work in OTS scenarios. Instead, the server must call **CoInitializeSecurity**, either explicitly or implicitly, and specify an ACL that includes the INTERACTIVE group SID and SYSTEM.

The following code example shows how to create a security descriptor (SD) with the INTERACTIVE group SID.

```
 BOOL GetAccessPermissionsForLUAServer(SECURITY_DESCRIPTOR **ppSD)
{
// Local call permissions to IU, SY
    LPWSTR lpszSDDL = L"O:BAG:BAD:(A;;0x3;;;IU)(A;;0x3;;;SY)";
    SECURITY_DESCRIPTOR *pSD;
    *ppSD = NULL;

    if (ConvertStringSecurityDescriptorToSecurityDescriptorW(lpszSDDL, SDDL_REVISION_1, (PSECURITY_DESCRIPTOR *)
    {
        *ppSD = pSD;
        return TRUE;
    }

    return FALSE;
}
```

The following code example shows how to call **CoInitializeSecurity** implicitly with the SD from the previous code example:

```
 // hKey is the HKCR\AppID\{GUID} key
BOOL SetAccessPermissions(HKEY hkey, PSECURITY_DESCRIPTOR pSD)
```

```
{
    BOOL bResult = FALSE;
    DWORD dwLen = GetSecurityDescriptorLength(pSD);
    LONG lResult;
    lResult = RegSetValueExA(hkey,
        "AccessPermission",
        0,
        REG_BINARY,
        (BYTE*)pSD,
        dwLen);
    if (lResult != ERROR_SUCCESS) goto done;
    bResult = TRUE;
done:
    return bResult;
}
```

The following code example shows how to call **CoInitializeSecurity** explicitly with the above SD:

```
 // Absolute SD values
PSECURITY_DESCRIPTOR pAbsSD = NULL;
DWORD AbsSdSize = 0;
PACL  pAbsAcl = NULL;
DWORD AbsAclSize = 0;
PACL  pAbsSacl = NULL;
DWORD AbsSaclSize = 0;
PSID  pAbsOwner = NULL;
DWORD AbsOwnerSize = 0;
PSID  pAbsGroup = NULL;
DWORD AbsGroupSize = 0;

MakeAbsoluteSD (
    pSD,
    pAbsSD,
    &AbsSdSize,
    pAbsAcl,
    &AbsAclSize,
    pAbsSacl,
    &AbsSaclSize,
    pAbsOwner,
    &AbsOwnerSize,
    pAbsGroup,
    &AbsGroupSize
);

if (ERROR_INSUFFICIENT_BUFFER == GetLastError())
{
```

```
    pAbsSD = (PSECURITY_DESCRIPTOR)LocalAlloc(LMEM_FIXED, AbsSdSize);
    pAbsAcl = (PACL)LocalAlloc(LMEM_FIXED, AbsAclSize);
    pAbsSacl = (PACL)LocalAlloc(LMEM_FIXED, AbsSaclSize);
    pAbsOwner = (PSID)LocalAlloc(LMEM_FIXED, AbsOwnerSize);
    pAbsGroup = (PSID)LocalAlloc(LMEM_FIXED, AbsGroupSize);

    if ( ! (pAbsSD && pAbsAcl && pAbsSacl && pAbsOwner && pAbsGroup))
    {
        hr = E_OUTOFMEMORY;
        goto Cleanup;
    }
    if ( ! MakeAbsoluteSD(
        pSD,
        pAbsSD,
        &AbsSdSize,
        pAbsAcl,
        &AbsAclSize,
        pAbsSacl,
        &AbsSaclSize,
        pAbsOwner,
        &AbsOwnerSize,
        pAbsGroup,
        &AbsGroupSize
        ))
    {
        hr = HRESULT_FROM_WIN32(GetLastError());
        goto Cleanup;
    }
}
else
{
    hr = HRESULT_FROM_WIN32(GetLastError());
    goto Cleanup;
}

// Call CoInitializeSecurity .
```

Windows Vista introduces the notion of *mandatory access labels* in security descriptors. The label dictates whether clients can get execute access to a COM object. The label is specified in the system access control list (SACL) portion of the security descriptor. In Windows Vista, COM supports the SYSTEM_MANDATORY_LABEL_NO_EXECUTE_UP label. SACLs in the COM permissions are ignored on operating systems prior to Windows Vista.

As of Windows Vista, dcomcnfg.exe does not support changing the integrity level (IL) in COM permissions. It must be set programmatically.

The following code example shows how to create a COM security descriptor with a label that allows launch/activation requests from all LOW IL clients. Note that the labels are valid for Launch/Activation and Call permissions. Thus, it is possible to write a COM server that disallows launch, activation or calls from clients with a certain IL. For more information about integrity levels, see the section "Understanding Windows Vista's Integrity Mechanism" in Understanding and Working in Protected Mode Internet Explorer.

```
 BOOL GetLaunchActPermissionsWithIL (SECURITY_DESCRIPTOR **ppSD)
{
// Allow World Local Launch/Activation permissions. Label the SD for LOW IL Execute UP
    LPWSTR lpszSDDL = L"O:BAG:BAD:(A;;0xb;;;WD)S:(ML;;NX;;;LW)";
    if (ConvertStringSecurityDescriptorToSecurityDescriptorW(lpszSDDL, SDDL_REVISION_1, (PSECURITY_DESCRIPTOR *)
    {
        *ppSD = pSD;
        return TRUE;
    }
}

BOOL SetLaunchActPermissions(HKEY hkey, PSECURITY_DESCRIPTOR pSD)
{

    BOOL bResult = FALSE;
    DWORD dwLen = GetSecurityDescriptorLength(pSD);
    LONG lResult;
    lResult = RegSetValueExA(hkey,
        "LaunchPermission",
        0,
        REG_BINARY,
        (BYTE*)pSD,
        dwLen);
    if (lResult != ERROR_SUCCESS) goto done;
    bResult = TRUE;
done:
    return bResult;
};
```

CoCreateInstance and Integrity Levels

The behavior of **CoCreateInstance** has changed in Windows Vista, to prevent Low IL clients from binding to COM servers by default. The server must explicitly allow such bindings by specifying the SACL. The changes to **CoCreateInstance** are as follows:

1. When launching a COM server process, the IL in the server process token is set to the client or server token IL, whichever is lower.
2. By default, COM will prevent Low IL clients from binding to running instances of any COM servers. To allow the bind, a COM server's Launch/Activation security descriptor must contain a SACL that specifies

the Low IL label (see the previous section for the sample code to create such a security descriptor).

Elevated Servers and ROT Registrations

If a COM server wishes to register in the running object table (ROT) and allow any client to access the registration, it must use the ROTFLAGS_ALLOWANYCLIENT flag. An "Activate As Activator" COM server cannot specify ROTFLAGS_ALLOWANYCLIENT because the DCOM service control manager (DCOMSCM) enforces a spoof check against this flag. Therefore, in Windows Vista, COM adds support for a new registry entry that allows the server to stipulate that its ROT registrations be made available to any client:

```
HKEY_LOCAL_MACHINE\Software\Classes\AppID
  {APPID}
     ROTFlags
```

The only valid value for this REG_DWORD entry is:

```
ROTREGFLAGS_ALLOWANYCLIENT 0x1
```

The entry must exist in the **HKEY_LOCAL_MACHINE** hive.

This entry provides an "Activate As Activator" server with the same functionality that ROTFLAGS_ALLOWANYCLIENT provides for a RunAs server.

Security in COM

Understanding and Working in Protected Mode Internet Explorer

---