Exercising the Firewall using C++

- Article
- 12/01/2010
- 8 minutes to read

The following code example exercises the Windows Firewall profile; displays the current profile, turns off the firewall, turns on the firewall, and adds an application.

```
/*
    Copyright (c) Microsoft Corporation

    SYNOPSIS

        Sample code for the Windows Firewall COM interface.
*/

#include <windows.h>
#include <crtdbg.h>
#include <netfw.h>
#include <objbase.h>
#include <oleauto.h>
#include <stdio.h>

#pragma comment( lib, "ole32.lib" )
#pragma comment( lib, "oleaut32.lib" )


HRESULT WindowsFirewallInitialize(OUT INetFwProfile** fwProfile)
{
    HRESULT hr = S_OK;
    INetFwMgr* fwMgr = NULL;
    INetFwPolicy* fwPolicy = NULL;

    _ASSERT(fwProfile != NULL);

    *fwProfile = NULL;

    // Create an instance of the firewall settings manager.
    hr = CoCreateInstance(
            __uuidof(NetFwMgr),
            NULL,
            CLSCTX_INPROC_SERVER,
            __uuidof(INetFwMgr),
            (void**)&fwMgr
            );
    if (FAILED(hr))
    {
        printf("CoCreateInstance failed: 0x%08lx\n", hr);
        goto error;
    }

    // Retrieve the local firewall policy.
    hr = fwMgr->get_LocalPolicy(&fwPolicy);
    if (FAILED(hr))
    {
        printf("get_LocalPolicy failed: 0x%08lx\n", hr);
        goto error;
    }

    // Retrieve the firewall profile currently in effect.
    hr = fwPolicy->get_CurrentProfile(fwProfile);
    if (FAILED(hr))
    {
```

```
        printf("get_CurrentProfile failed: 0x%08lx\n", hr);
        goto error;
    }

error:

    // Release the local firewall policy.
    if (fwPolicy != NULL)
    {
        fwPolicy->Release();
    }

    // Release the firewall settings manager.
    if (fwMgr != NULL)
    {
        fwMgr->Release();
    }

    return hr;
}


void WindowsFirewallCleanup(IN INetFwProfile* fwProfile)
{
    // Release the firewall profile.
    if (fwProfile != NULL)
    {
        fwProfile->Release();
    }
}


HRESULT WindowsFirewallIsOn(IN INetFwProfile* fwProfile, OUT BOOL* fwOn)
{
    HRESULT hr = S_OK;
    VARIANT_BOOL fwEnabled;

    _ASSERT(fwProfile != NULL);
    _ASSERT(fwOn != NULL);

    *fwOn = FALSE;

    // Get the current state of the firewall.
    hr = fwProfile->get_FirewallEnabled(&fwEnabled);
    if (FAILED(hr))
    {
        printf("get_FirewallEnabled failed: 0x%08lx\n", hr);
        goto error;
    }

    // Check to see if the firewall is on.
    if (fwEnabled != VARIANT_FALSE)
    {
        *fwOn = TRUE;
        printf("The firewall is on.\n");
```

```
    }
    else
    {
        printf("The firewall is off.\n");
    }

error:

    return hr;
}


HRESULT WindowsFirewallTurnOn(IN INetFwProfile* fwProfile)
{
    HRESULT hr = S_OK;
    BOOL fwOn;

    _ASSERT(fwProfile != NULL);

    // Check to see if the firewall is off.
    hr = WindowsFirewallIsOn(fwProfile, &fwOn);
    if (FAILED(hr))
    {
        printf("WindowsFirewallIsOn failed: 0x%08lx\n", hr);
        goto error;
    }

    // If it is, turn it on.
    if (!fwOn)
    {
        // Turn the firewall on.
        hr = fwProfile->put_FirewallEnabled(VARIANT_TRUE);
        if (FAILED(hr))
        {
            printf("put_FirewallEnabled failed: 0x%08lx\n", hr);
            goto error;
        }

        printf("The firewall is now on.\n");
    }

error:

    return hr;
}


HRESULT WindowsFirewallTurnOff(IN INetFwProfile* fwProfile)
{
    HRESULT hr = S_OK;
    BOOL fwOn;

    _ASSERT(fwProfile != NULL);

    // Check to see if the firewall is on.
```

```
    hr = WindowsFirewallIsOn(fwProfile, &fwOn);
    if (FAILED(hr))
    {
        printf("WindowsFirewallIsOn failed: 0x%08lx\n", hr);
        goto error;
    }

    // If it is, turn it off.
    if (fwOn)
    {
        // Turn the firewall off.
        hr = fwProfile->put_FirewallEnabled(VARIANT_FALSE);
        if (FAILED(hr))
        {
            printf("put_FirewallEnabled failed: 0x%08lx\n", hr);
            goto error;
        }

        printf("The firewall is now off.\n");
    }

error:

    return hr;
}


HRESULT WindowsFirewallAppIsEnabled(
            IN INetFwProfile* fwProfile,
            IN const wchar_t* fwProcessImageFileName,
            OUT BOOL* fwAppEnabled
            )
{
    HRESULT hr = S_OK;
    BSTR fwBstrProcessImageFileName = NULL;
    VARIANT_BOOL fwEnabled;
    INetFwAuthorizedApplication* fwApp = NULL;
    INetFwAuthorizedApplications* fwApps = NULL;

    _ASSERT(fwProfile != NULL);
    _ASSERT(fwProcessImageFileName != NULL);
    _ASSERT(fwAppEnabled != NULL);

    *fwAppEnabled = FALSE;

    // Retrieve the authorized application collection.
    hr = fwProfile->get_AuthorizedApplications(&fwApps);
    if (FAILED(hr))
    {
        printf("get_AuthorizedApplications failed: 0x%08lx\n", hr);
        goto error;
    }

    // Allocate a BSTR for the process image file name.
    fwBstrProcessImageFileName = SysAllocString(fwProcessImageFileName);
```

```
    if (fwBstrProcessImageFileName == NULL)
    {
        hr = E_OUTOFMEMORY;
        printf("SysAllocString failed: 0x%08lx\n", hr);
        goto error;
    }

    // Attempt to retrieve the authorized application.
    hr = fwApps->Item(fwBstrProcessImageFileName, &fwApp);
    if (SUCCEEDED(hr))
    {
        // Find out if the authorized application is enabled.
        hr = fwApp->get_Enabled(&fwEnabled);
        if (FAILED(hr))
        {
            printf("get_Enabled failed: 0x%08lx\n", hr);
            goto error;
        }

        if (fwEnabled != VARIANT_FALSE)
        {
            // The authorized application is enabled.
            *fwAppEnabled = TRUE;

            printf(
                "Authorized application %lS is enabled in the firewall.\n",
                fwProcessImageFileName
                );
        }
        else
        {
            printf(
                "Authorized application %lS is disabled in the firewall.\n",
                fwProcessImageFileName
                );
        }
    }
    else
    {
        // The authorized application was not in the collection.
        hr = S_OK;

        printf(
            "Authorized application %lS is disabled in the firewall.\n",
            fwProcessImageFileName
            );
    }

error:

    // Free the BSTR.
    SysFreeString(fwBstrProcessImageFileName);

    // Release the authorized application instance.
    if (fwApp != NULL)
```

```
    {
        fwApp->Release();
    }

    // Release the authorized application collection.
    if (fwApps != NULL)
    {
        fwApps->Release();
    }

    return hr;
}


HRESULT WindowsFirewallAddApp(
            IN INetFwProfile* fwProfile,
            IN const wchar_t* fwProcessImageFileName,
            IN const wchar_t* fwName
            )
{
    HRESULT hr = S_OK;
    BOOL fwAppEnabled;
    BSTR fwBstrName = NULL;
    BSTR fwBstrProcessImageFileName = NULL;
    INetFwAuthorizedApplication* fwApp = NULL;
    INetFwAuthorizedApplications* fwApps = NULL;

    _ASSERT(fwProfile != NULL);
    _ASSERT(fwProcessImageFileName != NULL);
    _ASSERT(fwName != NULL);

    // First check to see if the application is already authorized.
    hr = WindowsFirewallAppIsEnabled(
            fwProfile,
            fwProcessImageFileName,
            &fwAppEnabled
            );
    if (FAILED(hr))
    {
        printf("WindowsFirewallAppIsEnabled failed: 0x%08lx\n", hr);
        goto error;
    }

    // Only add the application if it isn't already authorized.
    if (!fwAppEnabled)
    {
        // Retrieve the authorized application collection.
        hr = fwProfile->get_AuthorizedApplications(&fwApps);
        if (FAILED(hr))
        {
            printf("get_AuthorizedApplications failed: 0x%08lx\n", hr);
            goto error;
        }

        // Create an instance of an authorized application.
```

```
hr = CoCreateInstance(
        __uuidof(NetFwAuthorizedApplication),
        NULL,
        CLSCTX_INPROC_SERVER,
        __uuidof(INetFwAuthorizedApplication),
        (void**)&fwApp
        );
if (FAILED(hr))
{
    printf("CoCreateInstance failed: 0x%08lx\n", hr);
    goto error;
}

// Allocate a BSTR for the process image file name.
fwBstrProcessImageFileName = SysAllocString(fwProcessImageFileName);
if (fwBstrProcessImageFileName == NULL)
{
    hr = E_OUTOFMEMORY;
    printf("SysAllocString failed: 0x%08lx\n", hr);
    goto error;
}

// Set the process image file name.
hr = fwApp->put_ProcessImageFileName(fwBstrProcessImageFileName);
if (FAILED(hr))
{
    printf("put_ProcessImageFileName failed: 0x%08lx\n", hr);
    goto error;
}

// Allocate a BSTR for the application friendly name.
fwBstrName = SysAllocString(fwName);
if (SysStringLen(fwBstrName) == 0)
{
    hr = E_OUTOFMEMORY;
    printf("SysAllocString failed: 0x%08lx\n", hr);
    goto error;
}

// Set the application friendly name.
hr = fwApp->put_Name(fwBstrName);
if (FAILED(hr))
{
    printf("put_Name failed: 0x%08lx\n", hr);
    goto error;
}

// Add the application to the collection.
hr = fwApps->Add(fwApp);
if (FAILED(hr))
{
    printf("Add failed: 0x%08lx\n", hr);
    goto error;
}
```

```
        printf(
            "Authorized application %lS is now enabled in the firewall.\n",
            fwProcessImageFileName
            );
    }

error:

    // Free the BSTRs.
    SysFreeString(fwBstrName);
    SysFreeString(fwBstrProcessImageFileName);

    // Release the authorized application instance.
    if (fwApp != NULL)
    {
        fwApp->Release();
    }

    // Release the authorized application collection.
    if (fwApps != NULL)
    {
        fwApps->Release();
    }

    return hr;
}


HRESULT WindowsFirewallPortIsEnabled(
            IN INetFwProfile* fwProfile,
            IN LONG portNumber,
            IN NET_FW_IP_PROTOCOL ipProtocol,
            OUT BOOL* fwPortEnabled
            )
{
    HRESULT hr = S_OK;
    VARIANT_BOOL fwEnabled;
    INetFwOpenPort* fwOpenPort = NULL;
    INetFwOpenPorts* fwOpenPorts = NULL;

    _ASSERT(fwProfile != NULL);
    _ASSERT(fwPortEnabled != NULL);

    *fwPortEnabled = FALSE;

    // Retrieve the globally open ports collection.
    hr = fwProfile->get_GloballyOpenPorts(&fwOpenPorts);
    if (FAILED(hr))
    {
        printf("get_GloballyOpenPorts failed: 0x%08lx\n", hr);
        goto error;
    }

    // Attempt to retrieve the globally open port.
    hr = fwOpenPorts->Item(portNumber, ipProtocol, &fwOpenPort);
```

```
    if (SUCCEEDED(hr))
    {
        // Find out if the globally open port is enabled.
        hr = fwOpenPort->get_Enabled(&fwEnabled);
        if (FAILED(hr))
        {
            printf("get_Enabled failed: 0x%08lx\n", hr);
            goto error;
        }

        if (fwEnabled != VARIANT_FALSE)
        {
            // The globally open port is enabled.
            *fwPortEnabled = TRUE;

            printf("Port %ld is open in the firewall.\n", portNumber);
        }
        else
        {
            printf("Port %ld is not open in the firewall.\n", portNumber);
        }
    }
    else
    {
        // The globally open port was not in the collection.
        hr = S_OK;

        printf("Port %ld is not open in the firewall.\n", portNumber);
    }

error:

    // Release the globally open port.
    if (fwOpenPort != NULL)
    {
        fwOpenPort->Release();
    }

    // Release the globally open ports collection.
    if (fwOpenPorts != NULL)
    {
        fwOpenPorts->Release();
    }

    return hr;
}


HRESULT WindowsFirewallPortAdd(
            IN INetFwProfile* fwProfile,
            IN LONG portNumber,
            IN NET_FW_IP_PROTOCOL ipProtocol,
            IN const wchar_t* name
            )
{
```

```
HRESULT hr = S_OK;
BOOL fwPortEnabled;
BSTR fwBstrName = NULL;
INetFwOpenPort* fwOpenPort = NULL;
INetFwOpenPorts* fwOpenPorts = NULL;

_ASSERT(fwProfile != NULL);
_ASSERT(name != NULL);

// First check to see if the port is already added.
hr = WindowsFirewallPortIsEnabled(
        fwProfile,
        portNumber,
        ipProtocol,
        &fwPortEnabled
        );
if (FAILED(hr))
{
    printf("WindowsFirewallPortIsEnabled failed: 0x%08lx\n", hr);
    goto error;
}

// Only add the port if it isn't already added.
if (!fwPortEnabled)
{
    // Retrieve the collection of globally open ports.
    hr = fwProfile->get_GloballyOpenPorts(&fwOpenPorts);
    if (FAILED(hr))
    {
        printf("get_GloballyOpenPorts failed: 0x%08lx\n", hr);
        goto error;
    }

    // Create an instance of an open port.
    hr = CoCreateInstance(
            __uuidof(NetFwOpenPort),
            NULL,
            CLSCTX_INPROC_SERVER,
            __uuidof(INetFwOpenPort),
            (void**)&fwOpenPort
            );
    if (FAILED(hr))
    {
        printf("CoCreateInstance failed: 0x%08lx\n", hr);
        goto error;
    }

    // Set the port number.
    hr = fwOpenPort->put_Port(portNumber);
    if (FAILED(hr))
    {
        printf("put_Port failed: 0x%08lx\n", hr);
        goto error;
    }
```

```
        // Set the IP protocol.
        hr = fwOpenPort->put_Protocol(ipProtocol);
        if (FAILED(hr))
        {
            printf("put_Protocol failed: 0x%08lx\n", hr);
            goto error;
        }

        // Allocate a BSTR for the friendly name of the port.
        fwBstrName = SysAllocString(name);
        if (SysStringLen(fwBstrName) == 0)
        {
            hr = E_OUTOFMEMORY;
            printf("SysAllocString failed: 0x%08lx\n", hr);
            goto error;
        }

        // Set the friendly name of the port.
        hr = fwOpenPort->put_Name(fwBstrName);
        if (FAILED(hr))
        {
            printf("put_Name failed: 0x%08lx\n", hr);
            goto error;
        }

        // Opens the port and adds it to the collection.
        hr = fwOpenPorts->Add(fwOpenPort);
        if (FAILED(hr))
        {
            printf("Add failed: 0x%08lx\n", hr);
            goto error;
        }

        printf("Port %ld is now open in the firewall.\n", portNumber);
    }

error:

    // Free the BSTR.
    SysFreeString(fwBstrName);

    // Release the open port instance.
    if (fwOpenPort != NULL)
    {
        fwOpenPort->Release();
    }

    // Release the globally open ports collection.
    if (fwOpenPorts != NULL)
    {
        fwOpenPorts->Release();
    }

    return hr;
}
```

```
int __cdecl wmain(int argc, wchar_t* argv[])
{
    HRESULT hr = S_OK;
    HRESULT comInit = E_FAIL;
    INetFwProfile* fwProfile = NULL;

    // Initialize COM.
    comInit = CoInitializeEx(
                0,
                COINIT_APARTMENTTHREADED | COINIT_DISABLE_OLE1DDE
                );

   // Ignore RPC_E_CHANGED_MODE; this just means that COM has already been
   // initialized with a different mode. Since we don't care what the mode is,
   // we'll just use the existing mode.
   if (comInit != RPC_E_CHANGED_MODE)
   {
        hr = comInit;
        if (FAILED(hr))
        {
            printf("CoInitializeEx failed: 0x%08lx\n", hr);
            goto error;
        }
   }

    // Retrieve the firewall profile currently in effect.
    hr = WindowsFirewallInitialize(&fwProfile);
    if (FAILED(hr))
    {
        printf("WindowsFirewallInitialize failed: 0x%08lx\n", hr);
        goto error;
    }

    // Turn off the firewall.
    hr = WindowsFirewallTurnOff(fwProfile);
    if (FAILED(hr))
    {
        printf("WindowsFirewallTurnOff failed: 0x%08lx\n", hr);
        goto error;
    }

    // Turn on the firewall.
    hr = WindowsFirewallTurnOn(fwProfile);
    if (FAILED(hr))
    {
        printf("WindowsFirewallTurnOn failed: 0x%08lx\n", hr);
        goto error;
    }

    // Add Windows Messenger to the authorized application collection.
    hr = WindowsFirewallAddApp(
            fwProfile,
            L"%ProgramFiles%\\Messenger\\msmsgs.exe",
```

```
            L"Windows Messenger"
            );
    if (FAILED(hr))
    {
        printf("WindowsFirewallAddApp failed: 0x%08lx\n", hr);
        goto error;
    }

    // Add TCP::80 to list of globally open ports.
    hr = WindowsFirewallPortAdd(fwProfile, 80, NET_FW_IP_PROTOCOL_TCP, L"WWW");
    if (FAILED(hr))
    {
        printf("WindowsFirewallPortAdd failed: 0x%08lx\n", hr);
        goto error;
    }

error:

    // Release the firewall profile.
    WindowsFirewallCleanup(fwProfile);

    // Uninitialize COM.
    if (SUCCEEDED(comInit))
    {
        CoUninitialize();
    }

    return 0;
}
```

Send comments about this topic to Microsoft

Build date: 11/30/2010