Sign in

gtworek / PSBits Public

Notifications Fork 525 Star 3.2k

Code Issues Pull requests Actions Projects Security Insights

PSBits / IFilter / Dll.cpp

346 lines (303 loc) · 8.86 KB

Code Blame Raw

```
// THIS CODE AND INFORMATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF
// ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO
// THE IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A
// PARTICULAR PURPOSE.
//
// Copyright (c) Microsoft Corporation. All rights reserved

// code is based on the https://docs.microsoft.com/en-us/samples/microsoft/windows-classic-samples/
// the original license may be seen at https://github.com/microsoft/Windows-classic-samples/blob/ma

#ifndef UNICODE
#error Unicode environment required. Some day, I will fix, if anyone needs it.
#endif

#include <Windows.h>
#include <new>
#include <Shlwapi.h>
#include <tchar.h>
#include <Psapi.h>

#define LOGFUNCTION TCHAR _strMsg[1024] = { 0 };\
                                        _stprintf_s(_strMsg, _countof(_strMsg), TEXT("[GTIFilter] %
                                        if (_tcslen(_strMsg) > 0)\
                                        {\
                                                OutputDebugString(_strMsg);\
                                        }
```

```

#define SZ_FILTERSAMPLE_CLSID L"{AF9925E4-9A8A-4927-994E-EFC65F2EC6DF}"
#define SZ_FILTERSAMPLE_HANDLER L"{80423D65-47FF-4C4E-B7BD-C91627824A93}"

#define USERNAME_LENGTH 512
#define DOMAINNAME_LENGTH 512

HRESULT CFilterSample_CreateInstance(REFIID riid, void** ppv);

// Handle to the DLL's module
HINSTANCE g_hInst = NULL;

// Module Ref count
long c_cRefModule = 0;


BOOL GetProcessUsername(HANDLE hProcess, LPTSTR lpUserName)
{
        HANDLE hToken = nullptr;
        PTOKEN_USER ptuTokenInformation = nullptr;
        DWORD dwTokenLength;
        DWORD dwUserNameLen = USERNAME_LENGTH;
        DWORD dwDomainNameLen = DOMAINNAME_LENGTH;
        TCHAR szUserName[USERNAME_LENGTH];
        TCHAR szDomainName[DOMAINNAME_LENGTH];
        SID_NAME_USE snuSidUse;
        TCHAR strNameBuf[USERNAME_LENGTH + 1 + DOMAINNAME_LENGTH] = {0};

        if (!OpenProcessToken(hProcess, TOKEN_QUERY, &hToken))
        {
                _tcscpy_s(strNameBuf, _countof(strNameBuf), TEXT("UNKNOWN"));
                lpUserName = strNameBuf;
                return FALSE;
        }

        GetTokenInformation(hToken, TokenUser, nullptr, 0, &dwTokenLength);
        ptuTokenInformation = static_cast<PTOKEN_USER>(LocalAlloc(LPTR, dwTokenLength));
        if (nullptr == ptuTokenInformation)
        {
                CloseHandle(hToken);
                _tcscpy_s(strNameBuf, _countof(strNameBuf), TEXT("UNKNOWN"));
                lpUserName = strNameBuf;
                return FALSE;
        }

        if (!GetTokenInformation(hToken, TokenUser, ptuTokenInformation, dwTokenLength, &dwTokenLen
```

```
        {
                CloseHandle(hToken);
                LocalFree(ptuTokenInformation);
                _tcscpy_s(strNameBuf, _countof(strNameBuf), TEXT("UNKNOWN"));
                lpUserName = strNameBuf;
                return FALSE;
        }

        if (!LookupAccountSid(nullptr, ptuTokenInformation->User.Sid, szUserName, &dwUserNameLen, s
                              &dwDomainNameLen, &snuSidUse))
        {
                CloseHandle(hToken);
                LocalFree(ptuTokenInformation);
                _tcscpy_s(strNameBuf, _countof(strNameBuf), TEXT("UNKNOWN"));
                lpUserName = strNameBuf;
                return FALSE;
        }

        _stprintf_s(strNameBuf, _countof(strNameBuf), TEXT("%s\\%s"), szDomainName, szUserName);
        _tcscpy_s(lpUserName, _countof(strNameBuf), strNameBuf);

        LocalFree(ptuTokenInformation);
        CloseHandle(hToken);
        return TRUE;
}


void DllAddRef()
{
        InterlockedIncrement(&c_cRefModule);
}

void DllRelease()
{
        InterlockedDecrement(&c_cRefModule);
}

class CClassFactory : public IClassFactory
{
public:
        CClassFactory(REFCLSID clsid) : m_cRef(1), m_clsid(clsid)
        {
                DllAddRef();
        }

        // IUnknown
```

```
            // IUnknown
```

```
        WCHAR szModuleName[MAX_PATH];

        if (!GetModuleFileNameW(g_hInst, szModuleName, ARRAYSIZE(szModuleName)))
        {
            hr = HRESULT_FROM_WIN32(GetLastError());
        }
        else
        {
            // List of registry entries we want to create
            const REGISTRY_ENTRY rgRegistryEntries[] =
            {
                // RootKey             KeyName
                {HKEY_LOCAL_MACHINE, L"Software\\Classes\\CLSID\\" SZ_FILTERSAMPLE_CLSID, N
                {
                    HKEY_LOCAL_MACHINE, L"Software\\Classes\\CLSID\\" SZ_FILTERSAMPLE_C
                    szModuleName
                },
                {
                    HKEY_LOCAL_MACHINE, L"Software\\Classes\\CLSID\\" SZ_FILTERSAMPLE_C
                    L"ThreadingModel", L"Both"
                },
                {
                    HKEY_LOCAL_MACHINE, L"Software\\Classes\\CLSID\\" SZ_FILTERSAMPLE_H
                    L"Filter Sample Persistent Handler"
                },
                {
                    HKEY_LOCAL_MACHINE,
                    L"Software\\Classes\\CLSID\\" SZ_FILTERSAMPLE_HANDLER L"\\Persister
                },
```

```
			{
				HKEY_LOCAL_MACHINE,
				L"Software\\Classes\\CLSID\\" SZ_FILTERSAMPLE_HANDLER
				L"\\PersistentAddinsRegistered\\{89BCB740-6119-101A-BCB7-00DD010655
			},
			{HKEY_LOCAL_MACHINE, L"Software\\Classes\\.filtersample", NULL, L"Filter Sa
			{HKEY_LOCAL_MACHINE, L"Software\\Classes\\.filtersample\\PersistentHandler"
		};
		hr = S_OK;
		for (int i = 0; i < ARRAYSIZE(rgRegistryEntries) && SUCCEEDED(hr); i++)
		{
			hr = CreateRegKeyAndSetValue(&rgRegistryEntries[i]);
		}
	}
	return hr;
}

// Unregisters this COM server
STDAPI DllUnregisterServer()
{
	LOGFUNCTION;
	HRESULT hr = S_OK;
	const PCWSTR rgpszKeys[] =
	{
		L"Software\\Classes\\CLSID\\" SZ_FILTERSAMPLE_CLSID,
		L"Software\\Classes\\CLSID\\" SZ_FILTERSAMPLE_HANDLER,
		L"Software\\Classes\\.filtersample"
	};

	// Delete the registry entries
	for (int i = 0; i < ARRAYSIZE(rgpszKeys) && SUCCEEDED(hr); i++)
	{
		DWORD dwError = RegDeleteTree(HKEY_LOCAL_MACHINE, rgpszKeys[i]);
		if (ERROR_FILE_NOT_FOUND == dwError)
		{
			// If the registry entry has already been deleted, say S_OK.
			hr = S_OK;
		}
		else
		{
			hr = HRESULT_FROM_WIN32(dwError);
		}
	}
	return hr;
}
```