MSEPX - Microsoft Security Essentials Process Exclusion Utility

deleted_twb, 26 May 2011 CPOL

4.81 (17 votes)

A utility that allows point and click additions to the Microsoft Security Essentials process exclusion list

This is an old version of the currently published article.

Download source code - 162.19 KB

Introduction

This article describes coding a C++ Win32 based utility application that allows the user to use point and click interface to add process source file names (the path and name of the executable) to the process exclusion list of Microsoft Security Essentials. It creates and uses a Windows service, uses multiple service threads, has a custom drag and drop, and does registry editing.

Please excuse the fact that the code is far from commercial quality. This was a two or three day hobby exercise for my personal amusement, but is a fairly well tested effort so it should work correctly. Much of the code was copied from third parties (most if not all of the drag and drop source is based on the

Info

Version	15
First Posted	21 Feb 2011
Views	15,053
Downloads	571
Bookmarked	31 times

Tagged as

C++
Windows
Win32
Dev
Intermediate
system
threads

Related Articles

SAPrefs - Netscape-like Preferences Dialog

XNA Snooker Club

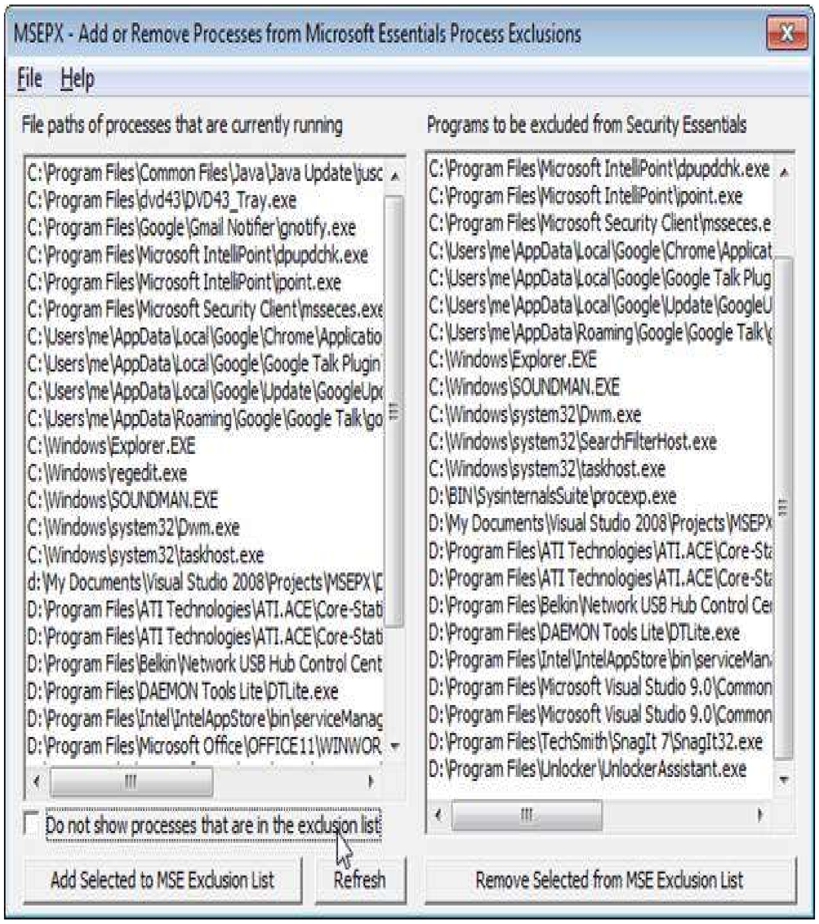
WPF: A* search

Windows 7 / VS2010 demo app

Window Tabs (WndTabs) Add-In for DevStudio

MSDN Magazine, October 2004 drag and drop code written by Paul DiLascia: Create Client Windows, Drag and Drop Between Listboxes) with the bulk of it based on examples from MSDN.

This program was written in C++ with Win32 libraries in Microsoft Visual Studio 8 and has been tested using Windows XP Professional and Windows 7 Ultimate.


MSEPX - Add or Remove Processes from Microsoft Essentials Process Exclusions

File Help

File paths of processes that are currently running

```
C:\Program Files\Common Files\Java\Java Update\jusc
C:\Program Files\dvd43\DVD43_Tray.exe
C:\Program Files\Google\Gmail Notifier\gnotify.exe
C:\Program Files\Microsoft IntelliPoint\dpupdchk.exe
C:\Program Files\Microsoft IntelliPoint\ipoint.exe
C:\Program Files\Microsoft Security Client\msseces.exe
C:\Users\me\AppData\Local\Google\Chrome\Applicatio
C:\Users\me\AppData\Local\Google\Google Talk Plugin
C:\Users\me\AppData\Local\Google\Update\GoogleUp
C:\Users\me\AppData\Roaming\Google\Google Talk\go
C:\Windows\Explorer.EXE
C:\Windows\regedit.exe
C:\Windows\SOUNDMAN.EXE
C:\Windows\system32\Dwm.exe
C:\Windows\system32\taskhost.exe
d:\My Documents\Visual Studio 2008\Projects\MSEPX\
D:\Program Files\ATI Technologies\ATI.ACE\Core-Stati
D:\Program Files\ATI Technologies\ATI.ACE\Core-Stati
D:\Program Files\Belkin\Network USB Hub Control Cent
D:\Program Files\DAEMON Tools Lite\DTLite.exe
D:\Program Files\Intel\IntelAppStore\bin\serviceManag
D:\Program Files\Microsoft Office\OFFICE11\WINWOR
```

Programs to be excluded from Security Essentials

```
C:\Program Files\Microsoft IntelliPoint\dpupdchk.exe
C:\Program Files\Microsoft IntelliPoint\ipoint.exe
C:\Program Files\Microsoft Security Client\msseces.e
C:\Users\me\AppData\Local\Google\Chrome\Applicat
C:\Users\me\AppData\Local\Google\Google Talk Plug
C:\Users\me\AppData\Local\Google\Update\GoogleU
C:\Users\me\AppData\Roaming\Google\Google Talk\
C:\Windows\Explorer.EXE
C:\Windows\SOUNDMAN.EXE
C:\Windows\system32\Dwm.exe
C:\Windows\system32\SearchFilterHost.exe
C:\Windows\system32\taskhost.exe
D:\BIN\SysinternalsSuite\procexp.exe
D:\My Documents\Visual Studio 2008\Projects\MSEPX
D:\Program Files\ATI Technologies\ATI.ACE\Core-Sta
D:\Program Files\ATI Technologies\ATI.ACE\Core-Sta
D:\Program Files\Belkin\Network USB Hub Control Ce
D:\Program Files\DAEMON Tools Lite\DTLite.exe
D:\Program Files\Intel\IntelAppStore\bin\serviceMan
D:\Program Files\Microsoft Visual Studio 9.0\Common
D:\Program Files\Microsoft Visual Studio 9.0\Common
D:\Program Files\TechSmith\SnagIt 7\SnagIt32.exe
D:\Program Files\Unlocker\UnlockerAssistant.exe
```

☐ Do not show processes that are in the exclusion list
Add Selected to MSE Exclusion List
Refresh
Remove Selected from MSE Exclusion List

The complete source code is included. The download link is at the top of the article.

Background

Microsoft Security Essentials (MSE) is a free and remarkably good malware protection system. However, the user interface is a little primitive, particularly the Excluded Processes interface that makes the user search for each individual process executable file when attempting to make MSE more efficient by adding processes to ignore into the registry location:

⊟ Collapse | Copy Code

Collapse | Copy Code

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft
Antimalware\Exclusions
```

The MSEPX (Microsoft Security Essentials Process Exclusion) utility which will be described here allows the user to select from a list of currently running process source file paths and add them to the MSE exclusion list using point and click and drag and drop techniques.

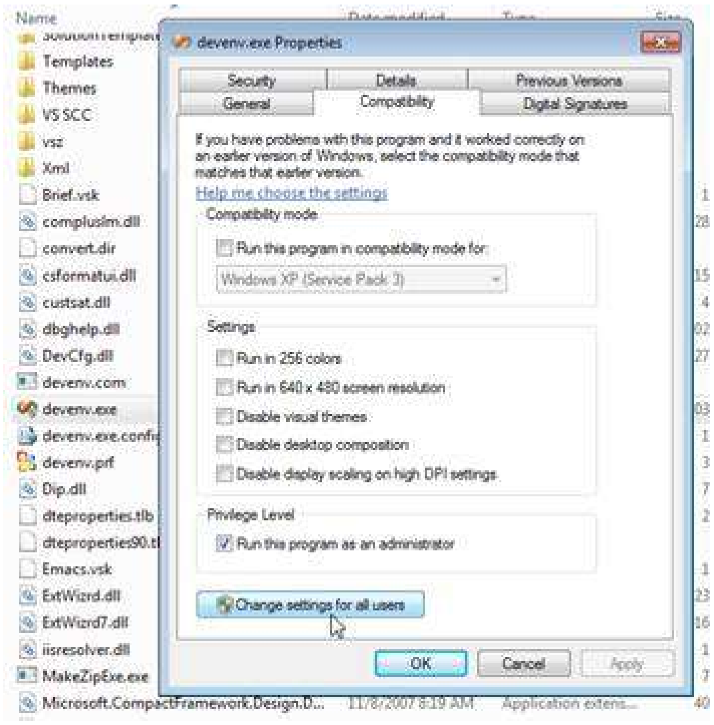
Enumerating processes and changing registry values is not complex, but security entries in the registry can have restricted permissions where the average user and even administrators can only read the values. Only the system user has complete access.

In order to become the system user, XP had security holes that could be exploited. The scheduled task trick: at 13:55 /interactive "*cmd.exe*" then killing and restarting explorer and the service control command launch method: `sc create testsvc binpath= "cmd /K start" type= own type= interact` could give you system access to do as you wished. These are clever but a little inelegant as a solution and Windows 7 eliminated these and other security loopholes so they are limited.

MSEPX began as an exercise in manipulating Windows security (using the fksec code as a base). The code would load the currently running processes and would change the registry location's permission settings allowing the user to alter the contents. This worked, but manipulating user permissions would be frowned upon in corporate or government environments.

To allow MSEPX to always be able to change the registry, a separate self loading Windows service was coded that MSEPX launched and communicated with using a named pipe inter-process communication. A Service, by being run by the local System account, can modify the registry, since the System account is allowed to do almost anything. Of course, when using Windows 7, the MSEPX program must be run as administrator in order to create a service.

The *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft Antimalware\Exclusions* is usually editable by the administrator so creating and running a service is not absolutely required. However, the code is designed work even if the service is not available so interacting with the service can be very easily edited out of the source. Note that in Windows 7, MSEPX has to be run as administrator in order to install or restart the service. This is also true when running it in Visual Studio. Make sure that *D:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\devenv.exe* has the privilege level set to run as administrator.



If you have problems with this program and it worked correctly on an earlier version of Windows, select the compatibility mode that matches that earlier version.

Help me choose the settings

Compatibility mode

☐ Run this program in compatibility mode for:

Windows XP (Service Pack 3)

Settings

☐ Run in 256 colors

☐ Run in 640 x 480 screen resolution

☐ Disable visual themes

☐ Disable desktop composition

☐ Disable display scaling on high DPI settings

Privilege Level

☑ Run this program as an administrator

Change settings for all users

OK Cancel Apply

Using the Code

MSEPX is basically two list boxes and two buttons so it is not a challenging user interface exercise. The interface coding was straightforward except I encountered a problem that the list boxes defaulted to only allowing a single selection and there was no multiple selection property to set to `true`. So, the *MSEPX.rc* file had to be edited directly and the `LBS_MULTIPLESEL` attribute added to both list boxes.

Collapse | Copy Code

Collapse | Copy Code

```
IDD_DLG_MAIN DIALOGEX 0, 0, 396, 239
STYLE DS_SETFONT | DS_MODALFRAME | DS_FIXEDSYS | WS_POPUP |
WS_CAPTION | WS_SYSMENU
CAPTION
"MSEPX - Add or Remove Processes from Microsoft Essentials
Process Exclusions"
MENU IDC_MSEPX
FONT 8, "MS Shell Dlg", 0, 0, 0x1
BEGIN
    LTEXT           "File paths of processes that are
currently running",
                IDC_STATIC,7,3,171,8
    LTEXT           "Programs to be excluded from Security
Essentials",
                IDC_STATIC,207,3,171,8
    LISTBOX
IDC_LIST_PROCESSES,7,16,190,191,LBS_SORT |
                LBS_MULTIPLESEL | LBS_NOINTEGRALHEIGHT |
LBS_EXTENDEDSEL |
                WS_VSCROLL | WS_HSCROLL | WS_TABSTOP
    LISTBOX
IDC_LIST_EXCLUDED,205,15,184,202,LBS_SORT |
                LBS_MULTIPLESEL | LBS_HASSTRINGS |
LBS_NOINTEGRALHEIGHT |
                LBS_EXTENDEDSEL | WS_VSCROLL | WS_HSCROLL |
WS_TABSTOP
    PUSHBUTTON      "Add Selected to MSE Exclusion List",
                IDC_BUTTON_MSEX,7,222,138,14
    PUSHBUTTON
"Refresh",IDC_BUTTON_REFRESH,151,222,45,14
    PUSHBUTTON      "Remove Selected from MSE Exclusion
List",
                IDC_BUTTON_MSEX_RMV,205,222,184,14
    CONTROL         "Do not show processes that are in the
exclusion list",
                IDC_CHECK_DONOT_SHOW_EXCLUDED,
                    "Button",BS_AUTOCHECKBOX |
WS_TABSTOP,7,208,183,10
END
```

Excluded Processes

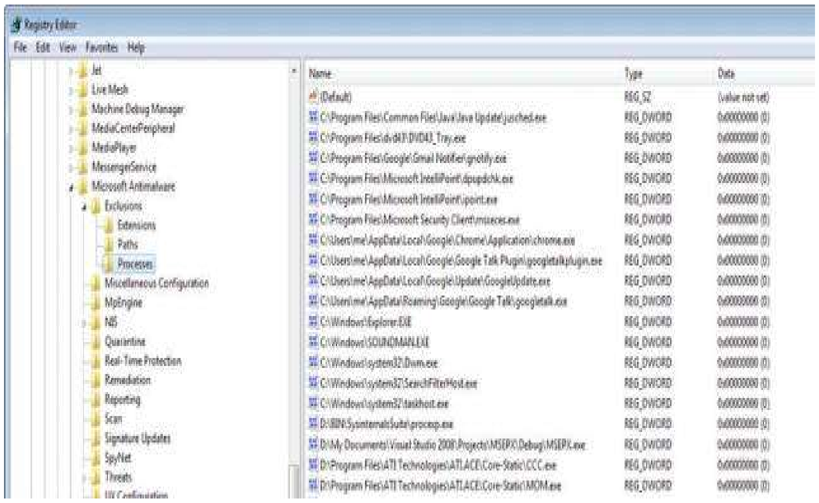
The MSEPX code starts first by enumerating the excluded processes in *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft Antimalware\Exclusions* and loading the excluded list box.

Collapse | Copy Code

Collapse | Copy Code

```
void SetExcludedList(HWND hListBox )
{
       int iExt = 0;  // for determining the horizontal
needed for list box
       TCHAR    achClass[MAX_PATH] = TEXT("");  // buffer
for class name
       DWORD    cchClassName = MAX_PATH;    // size of
class string
       DWORD    cSubKeys=0;                 // number of
subkeys
       DWORD    cbMaxSubKey;                // longest
subkey size
       DWORD    cchMaxClass;                // longest
class string
       DWORD    cValues;                // number of values
for key
       DWORD    cchMaxValue;            // longest value
name
       DWORD    cbMaxValueData;         // longest value
data
       DWORD    cbSecurityDescriptor;   // size of security
descriptor
       _FILETIME ftLastWriteTime;       // last write time
       DWORD i, retCode;
       TCHAR  achValue[MAX_VALUE_NAME];
       DWORD cchValue = MAX_VALUE_NAME;
       SendMessage(hListBox, LB_RESETCONTENT, 0, 0 );
       HKEY hKey;
       if( ERROR_SUCCESS == RegOpenKeyExW(
HKEY_LOCAL_MACHINE,
                                               MS_ESS_REG_XP,0,
KEY_READ, &hKey) )
       {
              // Get the class name and the value count.
              retCode = RegQueryInfoKey(
                     hKey,                  // key handle
                     achClass,              // buffer for
class name
                     &cchClassName,           // size of
class string
                     NULL,                    // reserved
                     &cSubKeys,               // number of
subkeys
                     &cbMaxSubKey,            // longest
subkey size
                     &cchMaxClass,            // longest
class string
                     &cValues,                // number of
values for this key
                     &cchMaxValue,            // longest
value name
                     &cbMaxValueData,         // longest
value data
                     &cbSecurityDescriptor,   // security
descriptor
                     &ftLastWriteTime);       // last write
time
              if (cValues)
              {
                     for(i=0, retCode = ERROR_SUCCESS ;
                         retCode == ERROR_SUCCESS && i <
cValues; i++)
                     {
                           cchValue = MAX_VALUE_NAME;
                           achValue[0] = '\0';
                           retCode = RegEnumValue(hKey, i,
                                  achValue,
                                  &cchValue,
                                  NULL,
                                  NULL,
                                  NULL,
                                  NULL);
                           if (retCode == ERROR_SUCCESS )
                           {
                                  SendMessage(hListBox,
LB_ADDSTRING, -1,
                                         (LPARAM)
(LPCSTR)achValue);
                                  int ilen =
GetTextLen(hMainDialog,achValue);
                                  if( iExt < ilen ) iExt =
ilen;
                           }
                     }
              }
       }
       SendMessage(hListBox, LB_SETHORIZONTALEXTENT, iExt,
0);
}
```

There are no subkeys below processes to deal with, so the code needed is very simple.



Name	Type	Data
(Default)	REG_SZ	(value not set)
C:\Program Files\Common Files\Java\Java Update\jusched.exe	REG_DWORD	0x00000000 (0)
C:\Program Files\dvd43\DVD43_Tray.exe	REG_DWORD	0x00000000 (0)
C:\Program Files\Google\Gmail Notifier\gnotify.exe	REG_DWORD	0x00000000 (0)
C:\Program Files\Microsoft IntelliPoint\dpupdchk.exe	REG_DWORD	0x00000000 (0)
C:\Program Files\Microsoft IntelliPoint\ipoint.exe	REG_DWORD	0x00000000 (0)
C:\Program Files\Microsoft Security Client\msseces.exe	REG_DWORD	0x00000000 (0)
C:\Users\me\AppData\Local\Google\Chrome\Application\chrome.exe	REG_DWORD	0x00000000 (0)
C:\Users\me\AppData\Local\Google\Google Talk Plugin\googletalkplugin.exe	REG_DWORD	0x00000000 (0)
C:\Users\me\AppData\Local\Google\Update\GoogleUpdate.exe	REG_DWORD	0x00000000 (0)
C:\Users\me\AppData\Roaming\Google\Google Talk\googletalk.exe	REG_DWORD	0x00000000 (0)
C:\Windows\Explorer.EXE	REG_DWORD	0x00000000 (0)
C:\Windows\SOUNDMAN.EXE	REG_DWORD	0x00000000 (0)
C:\Windows\system32\Dwm.exe	REG_DWORD	0x00000000 (0)
C:\Windows\system32\SearchFilterHost.exe	REG_DWORD	0x00000000 (0)
C:\Windows\system32\taskhost.exe	REG_DWORD	0x00000000 (0)
D:\BIN\SysinternalsSuite\procexp.exe	REG_DWORD	0x00000000 (0)
D:\My Documents\Visual Studio 2008\Projects\MSEPX\Debug\MSEPX.exe	REG_DWORD	0x00000000 (0)
D:\Program Files\ATI Technologies\ATI.ACE\Core-Static\CCC.exe	REG_DWORD	0x00000000 (0)
D:\Program Files\ATI Technologies\ATI.ACE\Core-Static\MOM.exe	REG_DWORD	0x00000000 (0)

Notice that there is a horizontal scroll bar for both list boxes. This is achieved with the last procedure call in the list populating code:

Collapse | Copy Code

Collapse | Copy Code

```
SendMessage(hListBox, LB_SETHORIZONTALEXTENT, iExt, 0);
```

This sets the list box to display up to the longest item string listed.

The processes added to the list are placed in the registry immediately. They are stored as `DWORD` values of zero with the path and name of the process file used as the variable name.

Running Processes

Next, running processes are enumerated and the source executable file paths are loaded to the currently running processes list box.

Collapse | Copy Code

Collapse | Copy Code

```
void SetProcessList(HWND hListBox, HWND hListBoxExclusions,
BOOL bFilter )
{
       HANDLE
hSnapShot=CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS |

TH32CS_SNAPMODULE32,0);
       //HANDLE
hSnapShot=CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS,0);
       PROCESSENTRY32* processInfo=new PROCESSENTRY32;
       processInfo->dwSize=sizeof(PROCESSENTRY32);

       SendMessage( hListBox, LB_RESETCONTENT, 0, 0 );
//clear the list
```

```
       wchar_t filepath[MAX_PATH] = {0};
       int iExt = 0;  // horizontal extent required for the
list box
       while(hSnapShot &&
Process32Next(hSnapShot,processInfo)!=FALSE)
       {
              DWORD RequiredAccess =
PROCESS_QUERY_INFORMATION | PROCESS_VM_READ;

              // Get a process handle
              HANDLE hProcess=OpenProcess
        ( RequiredAccess, TRUE, processInfo-
>th32ProcessID);
              if( hProcess )
              {
                     GetModuleFileNameEx(hProcess, NULL,
filepath, MAX_PATH );
                     if( *filepath )
                     {
                           if( !_tcsncmp(filepath, _T("\\??
\\ "), 4 ) )
                           {
                                  wchar_t *s = filepath+4;
                                  wchar_t *t = filepath;
                                  while( *s ) *t++ = *s++;
                                  ++*t = _T('\0');
                           }
                           else if( !_tcsnicmp(filepath,
_T("\\systemroot\\ "),12))
                           {
                                  wchar_t
rootpath[MAX_PATH] = {0};
                                  char * root = new
char[MAX_PATH];
                                  size_t sz = MAX_PATH;
                                  _dupenv_s(&root,
&sz,"systemroot");
                                  if(*root)
                                  {

MultiByteToWideChar(CP_ACP,MB_COMPOSITE,
                                                root,
strlen(root),
                                                rootpath,
sizeof(rootpath));
                                         wchar_t
path[MAX_PATH] = {0};

wcscpy_s(path,MAX_PATH,rootpath );

wcscat_s(path,MAX_PATH,filepath+11 );

wcscpy_s(filepath,MAX_PATH,path );
                                  }
                                  delete root;
                           }
                           BOOL bSkip = FALSE;
                           if( bFilter ) {
                                  bSkip = ( LB_ERR !=

SendMessage(hListBoxExclusions,

LB_FINDSTRINGEXACT, (WPARAM)-1,
                                                (LPARAM)
(LPCSTR) filepath ) );
                           }
                           if( !bSkip && ( LB_ERR ==
SendMessage(hListBox,
                                  LB_FINDSTRINGEXACT,
(WPARAM)-1, (LPARAM)(LPCSTR)
                                                                        filepath )) ) {
                                                            SendMessageW(hListBox,
LB_ADDSTRING, -1,
                                                                        (LPARAM)
(LPCSTR)filepath);
                                                            int ilen = GetTextLen(
hMainDialog, filepath);
                                                            if( iExt < ilen )
                                                                        iExt = ilen;
                                                }
                                    }
                                    CloseHandle( hProcess );
                        }
            }
            CloseHandle(hSnapShot);
            delete processInfo;
            SendMessage(hListBox, LB_SETHORIZONTALEXTENT, iExt,
0);
}
```

In order to get the file names, the *PSAPI.LIB* (Process Status API) library is required for calling the `GetModuleFileNameEx` function. If this program will be compiled to run on a 64 bit system, the flags in `CreateToolhelp32Snapshot` should be modified to `TH32CS_SNAPPROCESS | TH32CS_SNAPMODULE32` to include both 64 and 32 bit processes.

Some of the file paths start with \??\ (like *csrs.exe* and winlogon) or with \SystemRoot\ (like *smss.exe*) code is added to trim or translate these values. Also, some processes are duplicated so care has to be taken to avoid duplicates like services hosts (*svchost.exe*).

Drag and Drop

Instead of using COM for drag and drop which is really more appropriate for general program interaction, the drag and drop (DD) code simply allows copying between two list boxes. Most of the drag and drop source is based on the MSDN Magazine, October 2004 drag and drop code written by Paul DiLascia: Create Client Windows, Drag and Drop Between Listboxes with some added fixes and additions.

The DD code is currently limited to only allowing a single selection at a time and only drags from the process box to the exclusion box.

The DD code uses mouse movements and events and custom windows messages:

Collapse | Copy Code

Collapse | Copy Code

```
#define WM_DD_DRAGENTER (WM_USER + 0x0001)
#define WM_DD_DRAGDROP (WM_USER + 0x0002)
```

```
#define WM_DD_DRAGABORT (WM_USER + 0x0003)
#define WM_DD_DRAGOVER (WM_USER + 0x0004)
```

When the mouse is over a DD source object and the left mouse button is pressed, the DD code sends a `WM_DD_DRAGENTER` message that the main dialog code intercepts and creates a new `CDragDropText` class containing the selected list box item text. An image of the text is displayed as the item is being dragged into the destination and when over the destination area, in this case the exclusion list box, and the mouse button is released, the text is added to the list.

Note that the `CDragDropText::OnGetDragSize(HDC & dc)` function not only is used to calculate the drag image size but sets the image content using a call to `DrawText(dc, m_text, lstrlen(m_text)+1, &rc, DT_CALCRECT)` function which provides both the image size and setting of the content.

Other Controls

Note that the user interface code also handles double clicks (selects or deselects all items) and short cut keys (accelerators):

Collapse | Copy Code

Collapse | Copy Code

```
IDC_MSEPX ACCELERATORS
BEGIN
    "D",            IDC_BUTTON_MSEX_RMV,    VIRTKEY,
CONTROL, NOINVERT
    "R",            IDC_BUTTON_REFRESH,     VIRTKEY,
CONTROL, NOINVERT
    "A",            IDC_BUTTON_MSEX,        VIRTKEY,
CONTROL, NOINVERT
    "/",            IDM_ABOUT,              ASCII,
NOINVERT
    "?",            IDM_ABOUT,              ASCII,
NOINVERT
    VK_F1,          IDM_ABOUT,              VIRTKEY,
NOINVERT
    "Q",            IDM_EXIT,               VIRTKEY,
CONTROL, NOINVERT
    "X",            IDM_EXIT,               VIRTKEY,
CONTROL, NOINVERT
    "F",            IDC_CHECK_DONOT_SHOW_EXCLUDED, VIRTKEY,
CONTROL, NOINVERT
    VK_F5,          IDC_BUTTON_REFRESH,     VIRTKEY,
NOINVERT
END
```

There is also a refresh button that allows the user to launch new applications and then select refresh to include then in the process list.

MSEPXSVC Service

The `MSEPXSVC` program is coded as a self loading Windows service. Running as a service provides the code implicitly with System account privileges. `MSEPXSVC` uses a named pipe to communicate with the `MSEPX` client program and creates threads to process the commands that are sent through the pipe.

As mentioned, using a service in Windows 7 is not absolutely necessary to add to or delete from the MSE excluded process list. However, this did turn out to be a good example of coding a service, inter-process communications and using multi-threading to run tasks without delaying the user interface process.

An in depth introduction of services is available here: Introduction to Windows Service Applications.

Service Control

`MSEPX` first tries to start the service assuming the service was already installed. If not already an installed service, the restart fails and `MSEPX` launches the `MSEPXSVC` service code (included in the solution as a sub-project) and then waits for it to self install and again attempts to start the service. An argument (install) is sent to the `MSEPXSVC` process when calling the `CreateProcess` function. Without this argument, the service will not be installed. This is not really necessary since there are no other capabilities in the `MSEPXSVC` code. It was originally coded to allow debugging tests without having the service install and was left in case more debugging of the service was needed.

The `MSEPXSVC` code installs itself as the `RevValSvc` service. If looking at the running services with a task viewer like Sysinternals Process Explorer, it will be seen running as `MSEPXSVC`. In the Windows *services.msc* utility, it is seen as `RegValSvc` since this utility shows the services database rather than service processes.

In Windows 7, a service cannot be deleted and then recreated without restarting the machine. In Windows XP, this was possible to do. Because of this, `MSEPX` does not attempt to delete the service when it is closed. `MSPEX` only stops the `MSEPXSVC` `RegValSvc` service. However, if the users desire to completely remove the service, it can be done using the *SC.EXE* utility command:

Collapse | Copy Code

Collapse | Copy Code

```
SC delete RegValSvc
```

Event Logging and Debugging a Service

Debugging a service is a bit painful but can be done with a minimum of pain. Debugging is done using three methods: pre-install, post-install, and event logging.

Pre-Install Service Debugging

Debugging involves two separate processes. First, the code sections concerned with loading the service is debugged and then the service itself. To do the first, the service code is run as a separate project and this allows the developer to trace through the code that installs the process. The service code itself does not run unless created to run automatically which is not the case here.

Collapse | Copy Code

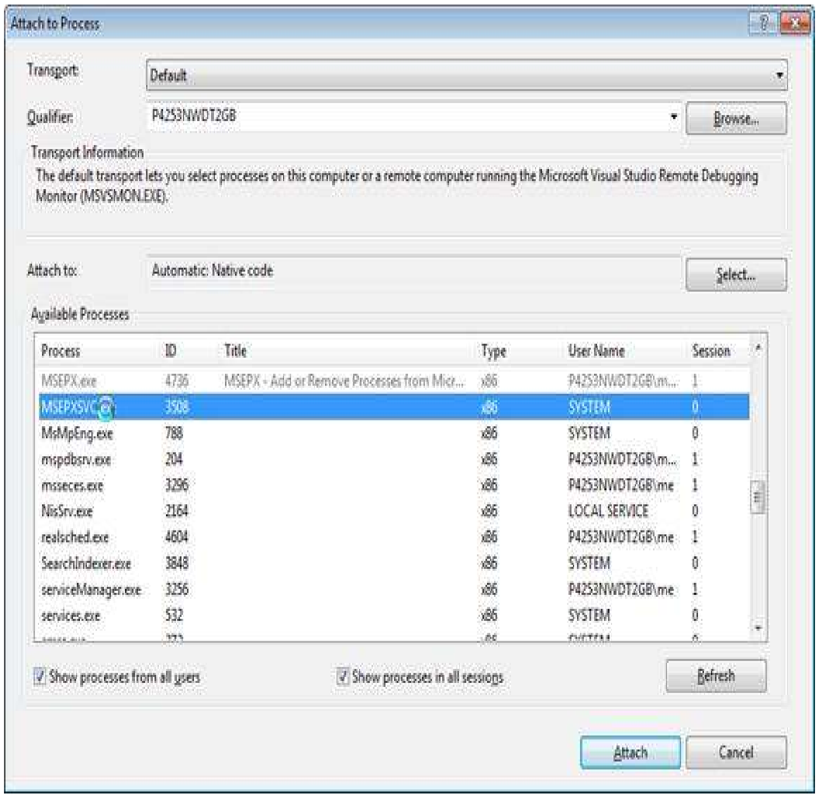
Collapse | Copy Code

```
// Create the service
schService = CreateService(
              schSCManager,              // SCM database
              SVCNAME,                   // name of service
              SVCNAME,                   // service name to
display
              SERVICE_ALL_ACCESS,        // desired access
              SERVICE_WIN32_OWN_PROCESS, // service type
              SERVICE_DEMAND_START,      // start type
              SERVICE_ERROR_NORMAL,      // error control
type
              szPath,                    // path to
service's binary
              NULL,                      // no load
ordering group
              NULL,                      // no tag
identifier
              NULL,                      // no dependencies
              NULL,                      // NULL means run
as LocalSystem account
              NULL);                     // no password
```

The above code in the `SvcInstall()` function in *svc.cpp* uses the `SERVICE_DEMAND_START` flag making the service manually started and not the `SERVICE_AUTO_START` flag. However, newly installed services do not run until the system is restarted or are started by the user or a program no matter which flag is used.

Post-Install Service Debugging

Next the process itself is debugged. To do this, a break point is set in the `MSEPX` code just after the point the service is started. At that point, select the Debug menu and Attach to process. Make sure the Show process from all users and Show processes from all sessions options are selected because `MSEPXSVC` is being run by System not the developer and will not be seen in the list otherwise.



Transport: Default

Qualifier: P4253NWDT2GB

Transport Information

The default transport lets you select processes on this computer or a remote computer running the Microsoft Visual Studio Remote Debugging Monitor (MSVSMON.EXE).

Attach to: Automatic: Native code

Available Processes

Process	ID	Title	Type	User Name	Session
MSEPX.exe	4736	MSEPX - Add or Remove Processes from Micr...	x86	P4253NWDT2GB\m...	1
MSEPXSVC.exe	3508		x86	SYSTEM	0
MsMpEng.exe	788		x86	SYSTEM	0
mspdbsrv.exe	204		x86	P4253NWDT2GB\m...	1
msseces.exe	3296		x86	P4253NWDT2GB\me	1
NisSrv.exe	2164		x86	LOCAL SERVICE	0
realsched.exe	4604		x86	P4253NWDT2GB\me	1
SearchIndexer.exe	3848		x86	SYSTEM	0
serviceManager.exe	3256		x86	P4253NWDT2GB\me	1
services.exe	532		x86	SYSTEM	0

[✓] Show processes from all users [✓] Show processes in all sessions

Event Logging

Even with the debugging capabilities of Visual Studio, it's nice to be able to keep an event log for a process. `MSEPXSRC` contains code that creates Windows application events in order to track the service execution. When an error is detected in a service, the code cannot stop and produce a nice little error message dialog for the user. A service can have no GUI based interaction with the user at all. Instead calls using the `ReportEvent` function are used to log information:

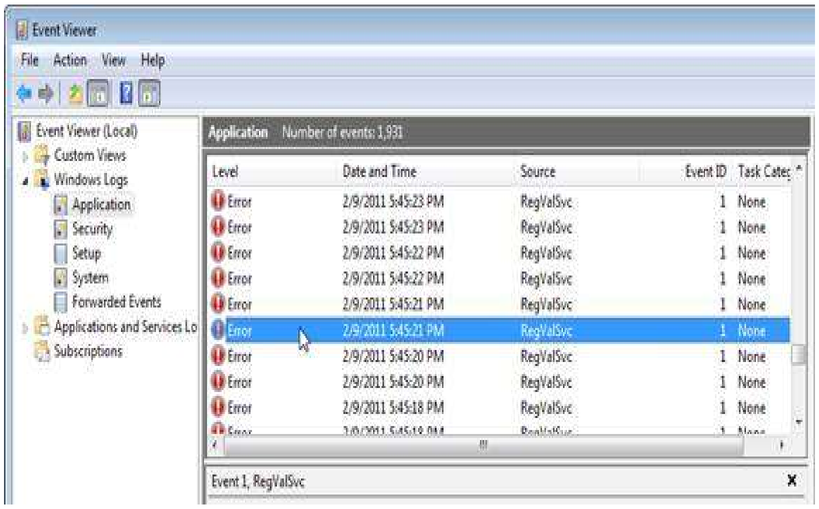
Collapse | Copy Code

Collapse | Copy Code

```
StringCchPrintf(mssg, MAX_SVC_MSSG_SZ,_T("Wrong arg count:
%d. Service Name: %s"),
                dwArgc, lpszArgv[0] );
SvcReportEvent(mssg);
 ...

VOID SvcReportEvent(LPTSTR szFunction) {
      HANDLE hEventSource;
```

```
       LPCTSTR lpszStrings[2];
       WCHAR Buffer[80];

       hEventSource = RegisterEventSource(NULL, SVCNAME);

       if( NULL != hEventSource )
       {
              StringCchPrintf(Buffer,80, TEXT("%s failed 
with %d"), 
                szFunction, GetLastError());

              lpszStrings[0] = SVCNAME;
              lpszStrings[1] = Buffer;

              ReportEvent(hEventSource,         // event 
log handle
                     EVENTLOG_ERROR_TYPE,   // event type
                     0,                     // event 
category
                     SVC_ERROR,             // event 
identifier
                     NULL,                  // no security 
identifier
                     2,                     // size of 
lpszStrings array
                     0,                     // no binary 
data
                     lpszStrings,           // array of 
strings
                     NULL);                 // no binary 
data

              DeregisterEventSource(hEventSource);
       }
}
```

Events are then viewed using the Windows event viewer.



Application Number of events: 1,931

Level	Date and Time	Source	Event ID	Task Categ
Error	2/9/2011 5:45:23 PM	RegValSvc	1	None
Error	2/9/2011 5:45:23 PM	RegValSvc	1	None
Error	2/9/2011 5:45:22 PM	RegValSvc	1	None
Error	2/9/2011 5:45:22 PM	RegValSvc	1	None
Error	2/9/2011 5:45:21 PM	RegValSvc	1	None
Error	2/9/2011 5:45:21 PM	RegValSvc	1	None
Error	2/9/2011 5:45:20 PM	RegValSvc	1	None
Error	2/9/2011 5:45:20 PM	RegValSvc	1	None
Error	2/9/2011 5:45:18 PM	RegValSvc	1	None

Event 1, RegValSvc

Details on coding event logging are covered in the MSDN article: Improve Manageability through Event Logging.

Inter-process Communications: Named Pipe

To communicate between the MSEPX client and the MSEPXSVC server running as the RegValSvc Windows service, a named pipe is used. Unlike UNIX and Linux, in Windows, there is no command line interface. Also unlike in UNIX, a named pipe in Windows is allocated in the root directory of the named pipe file system and mounted under the special path \\.*pipe*\ so a named pipe named XYZ has a path name of \\.*pipe**XYZ* and cannot be created on any writable file system. Much of the thread and pipe code used came from the MSDN example: Multithreaded Pipe Server. The MSEPXSVC service creates the named pipe \\.*pipe**MSEPX* and waits for the client to send a command through the pipe. When the command is received, the service processes it and sends back a message through the same pipe and then closes it.

Threading

Having the MSEPXSVC service create threads to process the registry edit commands from the MSEPX client has the advantage of being able to process commands from several different clients if necessary. The service has a `main` loop that sends status messages to the OS and listens for a service stop event. Once a `stop` event is encountered, the service stops until it is restarted by the MSEPX client. The service also creates a thread that handles processing the named pipe and that thread, when a command arrives through the pipe, then creates another thread that processes the command and adds to or deletes from the registry. The server is designed to take a serialized command from the pipe and construct a call to add or delete anything from the registry, not just Microsoft Security Essentials exclusions. When the pipe is read and the thread reading it creates the thread that will handle the command, the pipe handling thread closes the pipe and signals the service using an event that it has ended. The thread it created that handles the command is left to run and simply self destructs when finished.

Events

The MSEPXSVC service and the threads it creates both rely on another form of inter-process communication, the event object, to control how the code runs. The service code creates two event objects. The first is `ghSvcStopEvent` which is created in the service's main entry function and is used to monitor if a stop service signal was received from the service manager in the registered service control function. The second event object is `ThreadArgs.QuitEvent` which is created in the main service loop and passed to the pipe handling thread when created so that the thread can signal it has ended and a new pipe handling thread has to be created. Using the following code:

Collapse | Copy Code

Collapse | Copy Code

```
// Look for a stop service
if( WAIT_OBJECT_0 == WaitForSingleObject(ghSvcStopEvent,
200 )) //INFINITE) )
```

a thread can wait or loop until the event is detected and act on the event being set. The second parameter is the wait time in milliseconds which can be set to INFINITE (xFFFFFFFF) to wait forever for the event to be set by the `SetEvent(EventHandle)` command.

Points of Interest

Messing around with services, pipes, and threads made me appreciate the capabilities of Sysinternals Suite, particularly the Process Explorer utility. It's free and a must have for any Windows programmer.

History

This is version 1.0 of MSEPX created originally on November 12, 2010.

Minor edits and a change of category occurred May 25, 2011.

License

Share

EMAIL

About the Author

Comments and Discussions

Discussions posted for the Published version of this article. Posting a message here will take you to the publicly available article in order to continue your conversation in public.

Add a Comment or Question

Search Comments Go

Profile popups Spacing Relaxed ▼ Noise Medium ▼ Layout Normal ▼ Per page 25 ▼ Update

First Prev Next

Reclaiming the article	**TW_Burger**	**10mins ago**
How To Install Download	**Member 9041190**	**28-May-12 12:51**
Re: How To Install Download	twburger	28-May-12 16:50
My vote of 5	**Debojyoti Majumder**	**5-Apr-11 0:24**
Re: My vote of 5	twburger	28-Apr-11 14:06

Last Visit: 25-Oct-14 11:07 Last Update: 25-Oct-14 7:38 Refresh 1

General News Suggestion Question Bug Answer Joke Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

Permalink | Advertise | Privacy | Mobile
Web03 | 2.8.141022.2 | Last Updated 26 May 2011

Layout: fixed | fluid