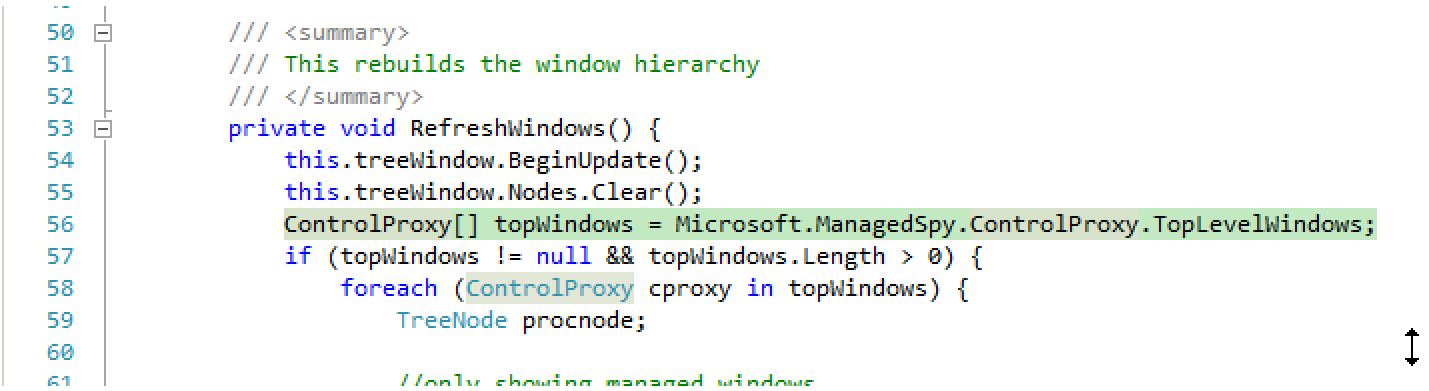
ManagedSpy - How ManagedSpyLib hooking works

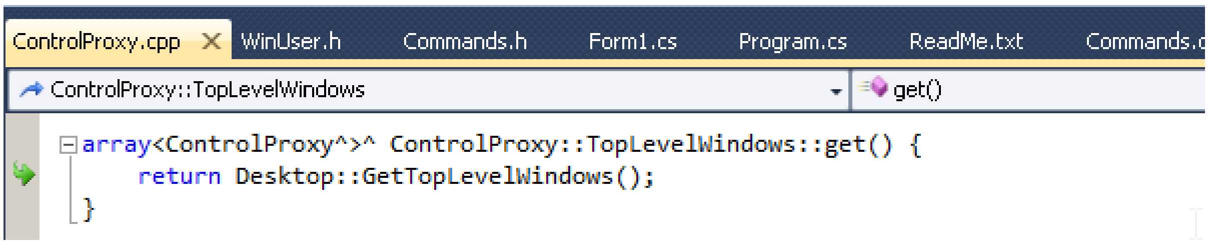
this is how the event hooking happens (on the remote process)

The C# ManagedSpy.exe code calls the ControlProxy.TopLevelWindows



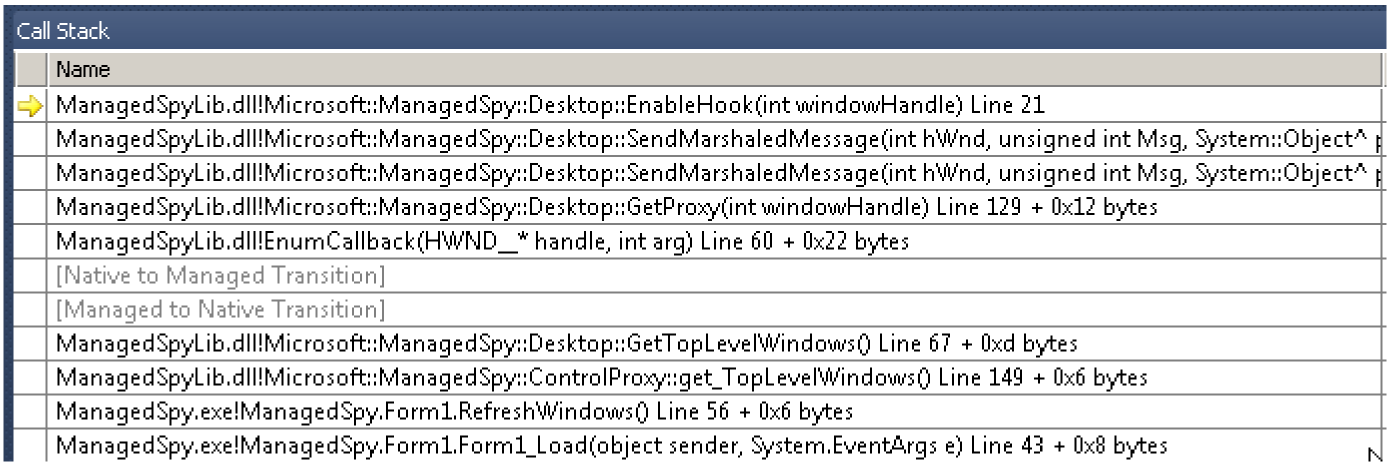
```
        /// <summary>
        /// This rebuilds the window hierarchy
        /// </summary>
        private void RefreshWindows() {
            this.treeWindow.BeginUpdate();
            this.treeWindow.Nodes.Clear();
            ControlProxy[] topWindows = Microsoft.ManagedSpy.ControlProxy.TopLevelWindows;
            if (topWindows != null && topWindows.Length > 0) {
                foreach (ControlProxy cproxy in topWindows) {
                    TreeNode procnode;

                    //only showing managed windows
```

which is in the C++ ManagedSpyLib.dll



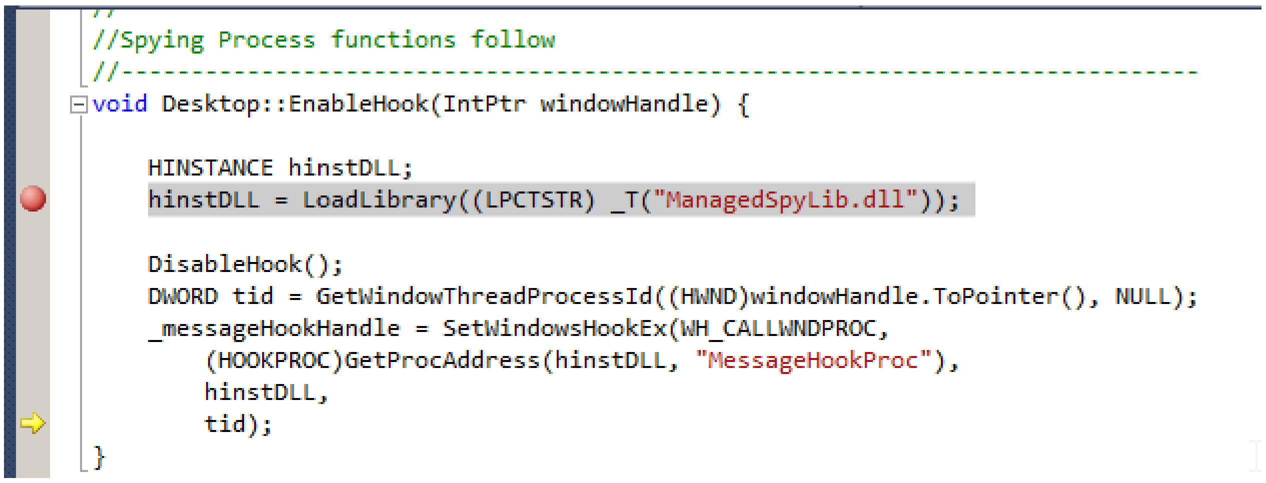
```
array<ControlProxy^>^ ControlProxy::TopLevelWindows::get() {
    return Desktop::GetTopLevelWindows();
}
```

eventually EnableHook is called



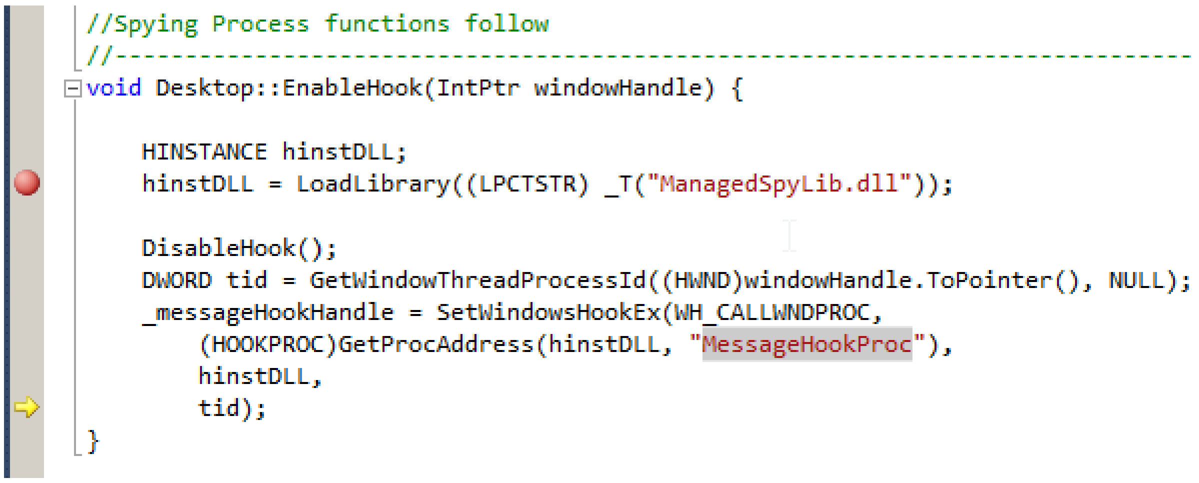
```
ManagedSpyLib.dll!Microsoft::ManagedSpy::Desktop::EnableHook(int windowHandle) Line 21
ManagedSpyLib.dll!Microsoft::ManagedSpy::Desktop::SendMarshaledMessage(int hWnd, unsigned int Msg, System::Object^
ManagedSpyLib.dll!Microsoft::ManagedSpy::Desktop::SendMarshaledMessage(int hWnd, unsigned int Msg, System::Object^
ManagedSpyLib.dll!Microsoft::ManagedSpy::Desktop::GetProxy(int windowHandle) Line 129 + 0x12 bytes
ManagedSpyLib.dll!EnumCallback(HWND__* handle, int arg) Line 60 + 0x22 bytes
[Native to Managed Transition]
[Managed to Native Transition]
ManagedSpyLib.dll!Microsoft::ManagedSpy::Desktop::GetTopLevelWindows() Line 67 + 0xd bytes
ManagedSpyLib.dll!Microsoft::ManagedSpy::ControlProxy::get_TopLevelWindows() Line 149 + 0x6 bytes
ManagedSpy.exe!ManagedSpy.Form1.RefreshWindows() Line 56 + 0x6 bytes
ManagedSpy.exe!ManagedSpy.Form1.Form1_Load(object sender, System.EventArgs e) Line 43 + 0x8 bytes
```

which will get a pointer to the loaded ManagedSpyLib.dll library



```
//Spying Process functions follow
//---------------------------------------------------------------------------
void Desktop::EnableHook(IntPtr windowHandle) {

    HINSTANCE hinstDLL;
    hinstDLL = LoadLibrary((LPCTSTR) _T("ManagedSpyLib.dll"));

    DisableHook();
    DWORD tid = GetWindowThreadProcessId((HWND)windowHandle.ToPointer(), NULL);
    _messageHookHandle = SetWindowsHookEx(WH_CALLWNDPROC,
        (HOOKPROC)GetProcAddress(hinstDLL, "MessageHookProc"),
        hinstDLL,
        tid);
}
```

in order to set a `WH_CALLWNDPROC` **to the** `MessageHookProc` **method (which is a callback that will be called before they are sent to the destination window)**



```
//Spying Process functions follow
//---------------------------------------------------------------------------
void Desktop::EnableHook(IntPtr windowHandle) {

    HINSTANCE hinstDLL;
    hinstDLL = LoadLibrary((LPCTSTR) _T("ManagedSpyLib.dll"));

    DisableHook();
    DWORD tid = GetWindowThreadProcessId((HWND)windowHandle.ToPointer(), NULL);
    _messageHookHandle = SetWindowsHookEx(WH_CALLWNDPROC,
        (HOOKPROC)GetProcAddress(hinstDLL, "MessageHookProc"),
        hinstDLL,
        tid);
}
```

Here is MSDN description (http://msdn.microsoft.com/en-gb/library/windows/desktop/ms644990(v=vs.85).aspx):

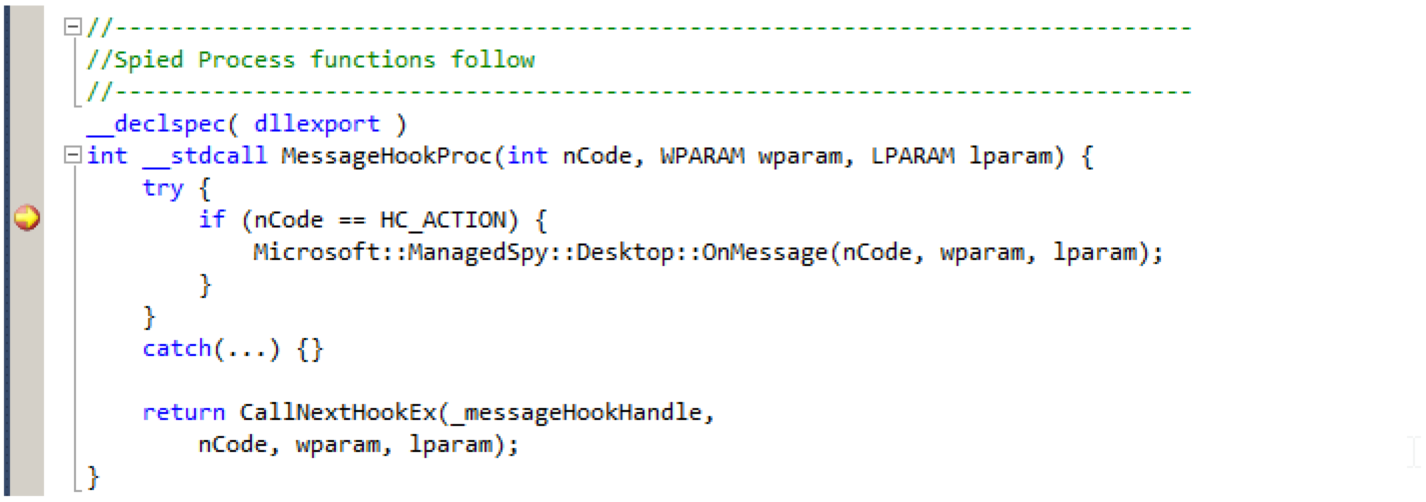
idHook [in]

Type: **int**

The type of hook procedure to be installed. This parameter can be one of the following values.

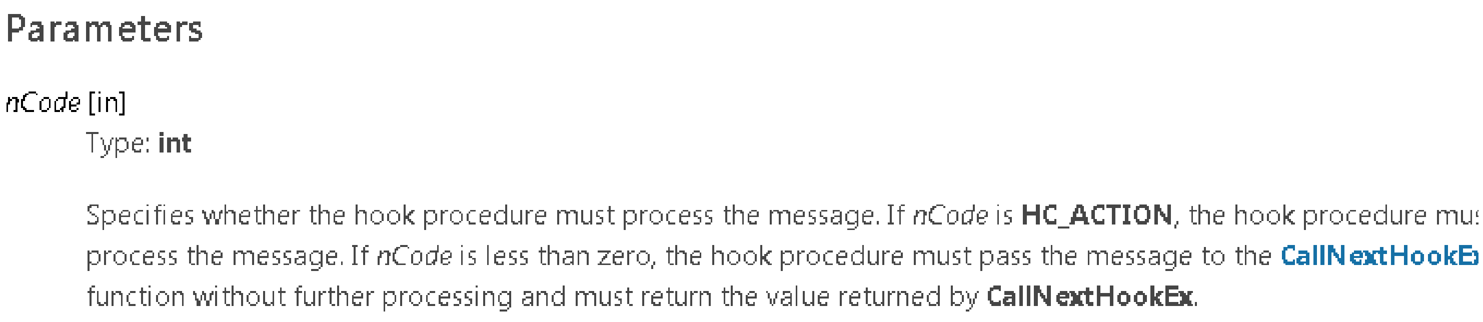
Value	Meaning
WH_CALLWNDPROC 4	Installs a hook procedure that monitors messages before the system sends them to the destination window procedure. For more information, see the **CallWndProc** hook procedure.

Here is the MessageHookProc being called on an message



```
//------------------------------------------------------------------------
//Spied Process functions follow
//------------------------------------------------------------------------
__declspec( dllexport )
int __stdcall MessageHookProc(int nCode, WPARAM wparam, LPARAM lparam) {
    try {
        if (nCode == HC_ACTION) {
            Microsoft::ManagedSpy::Desktop::OnMessage(nCode, wparam, lparam);
        }
    }
    catch(...) {}

    return CallNextHookEx(_messageHookHandle,
        nCode, wparam, lparam);
}
```

The HC_ACTION menas that the message should be processed (http://msdn.microsoft.com/en-gb/library/windows/desktop/ms644975(v=vs.85).aspx):



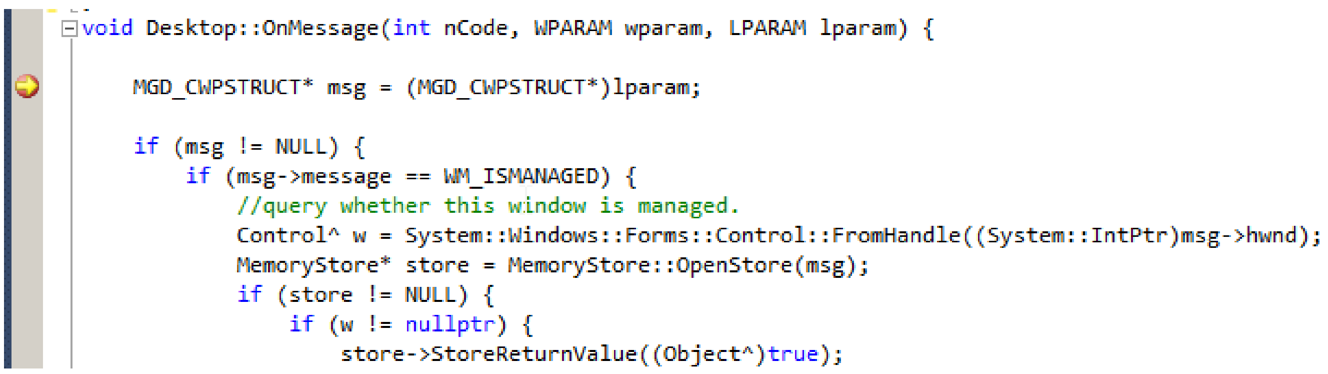
Parameters

nCode [in]

Type: **int**

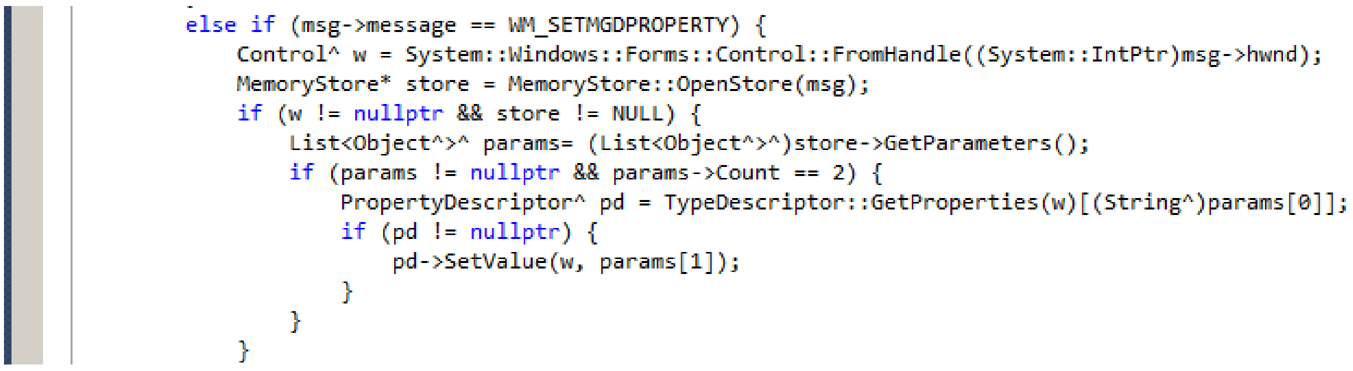
Specifies whether the hook procedure must process the message. If *nCode* is **HC_ACTION**, the hook procedure mus process the message. If *nCode* is less than zero, the hook procedure must pass the message to the **CallNextHookE** function without further processing and must return the value returned by **CallNextHookEx**.

The OnMessage is called



```
void Desktop::OnMessage(int nCode, WPARAM wparam, LPARAM lparam) {

    MGD_CWPSTRUCT* msg = (MGD_CWPSTRUCT*)lparam;

    if (msg != NULL) {
        if (msg->message == WM_ISMANAGED) {
            //query whether this window is managed.
            Control^ w = System::Windows::Forms::Control::FromHandle((System::IntPtr)msg->hwnd);
            MemoryStore* store = MemoryStore::OpenStore(msg);
            if (store != NULL) {
                if (w != nullptr) {
                    store->StoreReturnValue((Object^)true);
```

which has a big if-else loop to see if we the current message should be handled



```
        else if (msg->message == WM_SETMGDPROPERTY) {
            Control^ w = System::Windows::Forms::Control::FromHandle((System::IntPtr)msg->hwnd);
            MemoryStore* store = MemoryStore::OpenStore(msg);
            if (w != nullptr && store != NULL) {
                List<Object^>^ params= (List<Object^>^)store->GetParameters();
                if (params != nullptr && params->Count == 2) {
                    PropertyDescriptor^ pd = TypeDescriptor::GetProperties(w)[(String^)params[0]];
                    if (pd != nullptr) {
                        pd->SetValue(w, params[1]);
                    }
                }
            }
```