KOMODIA'S ADVANCED INSTALLER MANUAL
By Barak Weichselbaum

Introduction
This manual covers all the options supported by Komodia's advanced installer. Throughout this manual, the terms "installer" and "RegisterLSP" will refer to Komodia's advanced installer. It's important to keep in mind that although a small subset of the options is found in the default Microsoft SDK installer, the core functionality described here is unique to Komodia's installer.

This manual briefly discusses some LSP technical issues in order to give a reader unfamiliar with LSP a firmer grasp of the subject. However, this cannot substitute for a full understanding. A good place to start is my book **LSP: The Complete Guide** by Barak Weichselbaum.

Naming and licensing
When RegisterLSP is licensed, it is for a single LSP name only. That is why we have omitted the –n flag which exists in the Microsoft installer and allows to change the name of the LSP.

IFS and Non-IFS LSP
Before delving into the mechanics of the actual install process, we need to briefly review the differences between IFS and non IFS LSP (for more in-depth coverage of this topic and of the structure of the LSP stack, you can refer to my book.)

IFS LSP uses an actual OS I/O handle per socket, which means that handles passed from the LSP to the upper layer or application do not need translation.

Non-IFS LSP allocates its own handle per socket, which means that handles passed from the LSP to the upper layer of application must be translated by the LSP when it passes all the commands to its lower-layer LSP or base provider.

It is important to understand that from an installer point of view, each LSP needs to be installed differently. Non-IFS LSP can be layered into any position in the stack. IFS LSP **must** be installed first after the base providers or after another IFS-based LSP. However, IFS LSP cannot be installed after a Non-IFS LSP. Another important point is that the installer needs to set a special flag to denote that the IFS LSP is installed in the stack.

LSP dummy entry
Another mechanism to discuss up before we get to the fun part is the usage of LSP dummy entries. LSP is installed on top of multiple protocols, usually TCP and UDP; therefore, the installer creates a dummy entry with unique GUID different than that of the LSP. This entry is then named using the next layer's name plus the LSP, separated by the "over" keyword. The name is created for easy understanding, and has no technical significance. (Note: some programmers *do* assign meaning to the dummy entry's name, but this is wrong practice, as it is not defined as a crucial standard.)
A name can look like this:

```
TestLSP over [MSAFD Tcpip [TCP/IP]]
```

This dummy entry is used to create a chain, and inside the chain's entries it points to the actual LSP. For example, let's inspect this chain:

```
LSP Installed under 2 GUIDs
          {D7E3B4EA-D34A-4FE7-BF66-02B5E0A38BBF}
          {4EEC5B94-2F44-4D3C-9AB1-9D13658880C8}
      Layer 1021  "TestLSP over [MSAFD Tcpip [TCP/IP]]"
            Chain 2 [ 1020 1001 ]
      Layer 1022  "TestLSP over [MSAFD Tcpip [UDP/IP]]"
   Chain 2 [ 1020 1002 ]
```

and the following layered LSP output (later in the manual we discuss how to get these lists):

```
Winsock 32-bit Catalog:
=======================
1021 - TestLSP over [MSAFD Tcpip [TCP/IP]]
1022 - TestLSP over [MSAFD Tcpip [UDP/IP]]
1001 - MSAFD Tcpip [TCP/IP]
1002 - MSAFD Tcpip [UDP/IP]
1003 - MSAFD Tcpip [RAW/IP]
1020 - TestLSP
```

As the output shows, the LSP is installed using two GUIDs, one for each dummy entry (TCP and UDP), and each chain (bold and underlined) is composed of LSP ID 1020, which is our LSP, together with the relevant base provider ID. The ID for TCP is 1001 (the base provider for TCP, as seen in the list,) and the ID for UDP is 1002.

Autopilot

Syntax

The autopilot is the most common usage of the installer when trying to actually install an LSP. It analyzes the stack and determines the best course of action on the fly.

Installation syntax for a Non-IFS LSP install:

```
RegisterLSP -b -d LSPName
```

Installation syntax for an IFS LSP install:

```
RegisterLSP -bi -d LSPName
```

Important:

- When RegisterLSP and the LSP are in the same directory, there is no need to specify the directory name
- When the LSP path includes spaces, the path must be encapsulated in double quotes
- The installed LSP is copied to the System32 directory, so it's okay to delete the original LSP file after installation

Practical examples

Installing a Non-IFS LSP named *nonifslsp.dll* located at c:\lsp when the RegisterLSP is located elsewhere:

```
RegisterLSP –b –d c:\lsp\nonifslsp.dll
```

Installing a Non-IFS LSP named *nonifslsp.dll* located at “c:\test lsp” when the RegisterLSP is located elsewhere:

```
RegisterLSP –b –d “c:\test lsp\nonifslsp.dll”
```

Installing a Non-IFS LSP named *nonifslsp.dll* when the RegisterLSP is located in the same directory as the LSP:

```
RegisterLSP –b –d nonifslsp.dll
```

When installing an IFS LSP all the examples remain the same, but the flag –b is replaced with –bi (and, of course, the LSP name should be *ifslsp.dll*). Thus, the first example would become:

```
RegisterLSP –bi –d c:\lsp\ifslsp.dll
```

Vista issues

Under Vista, the autopilot detects whether it’s running in UAC elevated mode or not. If it isn’t, the installation process will stop with an error message saying it cannot install without UAC privileges.

UAC elevated mode is required for all install and update operations. The data collecting function doesn’t require UAC elevated mode.

If LSP already exists

When the LSP already exists (RegisterLSP detects a match based on the LSP GUID), the RegisterLSP compares the checksum of the source LSP with the checksum of the existing LSP. If the checksums are identical and the required installation strategy is the same, the RegisterLSP exists saying there is nothing to update.

In case the files are different, it is an update install, and the RegisterLSP replaces the existing LSP with the source LSP. In case any temporary files are created, they are queued to be deleted after the next reboot (rebooting is not required for the install process.)

In case the existing installation strategy differs from the new required strategy, the installer updates the LSP installation to the new strategy and copies the source file to the System32 directory if necessary.

How the autopilot works

Autopilot first analyzes the current stack condition and has a different strategy for each condition:

No LSP installed:

1. The installer copies the original LSP to System32
2. The installer performs the LSP install

LSP installed is the same LSP we are trying to install:

1. The installer checks if the source LSP and the existing LSP match:
    a. If they do, the installer continues to do a strategy check
    b. If they do not match, the installer replaces the existing LSP with the new one. Any temporary files created in the process are queued to be deleted after a reboot
2. The installer checks if the existing installation strategy is the required strategy:
    a. If they match, the installer exits
    b. If they don't match, the installer updates the existing strategy to match the new required strategy

The LSP we are trying to install doesn't exist, but another LSP does exist:

1. If the LSP we are trying to install is an IFS LSP, then it must be installed immediately after the base providers or after the last IFS LSP (experience shows that it is very difficult to stabilize a stack with more then two LSP installed)
2. If the LSP we are trying to install is a Non-IFS LSP, then the installer tries to install it as follows:
    a. The installer tries to install the new LSP to be first in the chain, which means that it processes every winsock command first (in manual installation, this is done by adding the –t flag. Refer to the Manual Installation section for more information.) In case of failure to install (the installer tries to connect to the Internet,) the installer tries the next strategy
    b. The installer tries to install the new LSP to be after the base provider, while keeping the existing installed LSP first in the chain (some LSPs require this due to architectural design or bad programming)
    c. There are rare LSPs that cannot have another LSP in the chain. In these cases, new LSPs must be installed over the base provider (this is called bypass mode), or the old "bad" LSP must be removed. Bypass mode is never chosen by the autopilot, but can only be preprogrammed using a scenario file.

Uninstalling an LSP

Uninstall all installed LSPs

Using the –f flag, the installer removes all user LSPs and leaves only base providers installed in the stack:

```
RegisterLSP –f
```

Uninstalling a specific LSP using its ID

Let us suppose that we have the following LSP stack:

```
Winsock 32-bit Catalog:
```

```
======================
1021 - TestLSP over [MSAFD Tcpip [TCP/IP]]
1022 - TestLSP over [MSAFD Tcpip [UDP/IP]]
1001 - MSAFD Tcpip [TCP/IP]
1002 - MSAFD Tcpip [UDP/IP]
1003 - MSAFD Tcpip [RAW/IP]
1004 - RSVP UDP Service Provider
1005 - RSVP TCP Service Provider
1012  -  MSAFD  NetBIOS  [\Device\NetBT_Tcpip_{0E41ADAF-748D-
4FE4-A76F-F164642EEE87}] SEQPACKET 3
1013  -  MSAFD  NetBIOS  [\Device\NetBT_Tcpip_{0E41ADAF-748D-
4FE4-A76F-F164642EEE87}] DATAGRAM 3
1014  -  MSAFD  NetBIOS  [\Device\NetBT_Tcpip_{B8E72AD6-0B84-
4188-8841-52340436256E}] SEQPACKET 0
1015  -  MSAFD  NetBIOS  [\Device\NetBT_Tcpip_{B8E72AD6-0B84-
4188-8841-52340436256E}] DATAGRAM 0
1016  -  MSAFD  NetBIOS  [\Device\NetBT_Tcpip_{A0299C89-898F-
4CF1-BCDC-1EE2550DFC55}] SEQPACKET 1
1017  -  MSAFD  NetBIOS  [\Device\NetBT_Tcpip_{A0299C89-898F-
4CF1-BCDC-1EE2550DFC55}] DATAGRAM 1
1018  -  MSAFD  NetBIOS  [\Device\NetBT_Tcpip_{9593A3E5-5C81-
4534-9160-0D9376807B68}] SEQPACKET 2
1019  -  MSAFD  NetBIOS  [\Device\NetBT_Tcpip_{9593A3E5-5C81-
4534-9160-0D9376807B68}] DATAGRAM 2
1020 - TestLSP
```

We want to uninstall the LSP “TestLSP”. We do it using its ID, 1020:

```
RegisterLSP –r 1020
```

Uninstalling a specific LSP using its name

Using the stack from the previous example, in order to remove the LSP “TestLSP,” we use the following syntax:

```
RegisterLSP –q TestLSP
```

In case the LSP name contains spaces, it must be encapsulated with double quotes. For an LSP named “My company LSP,” we would therefore use the following syntax:

```
RegisterLSP –q “My company LSP”
```

Disabling file removal on reboot

When uninstalling a specific LSP using –q or –r flags, the installer queues the file to be deleted from the System32 directory after the next reboot. To disable file deletion, the flag –s can be used, for example:

```
RegisterLSP –q “My company LSP” –s
```

Displaying LSP stack information

The structure of the stack is used to determine several important facts about the current stack condition:

- Which LSPs are installed and which are not
- Is an installed LSP IFS or Non-IFS
- The order of the LSPs inside a chain (for further information on LSP chains, you can refer to my book)
- How many chains exist

Basic output

The most basic way to view installed LSPs is:

```
RegisterLSP –p
```

A sample output would look like the stack displayed in the section "Uninstalling a specific LSP using its ID", above.

From this output we can extract:

- A list of base providers and their IDs
- A list of custom installed LSPs and their IDs
- The location of the custom LSPs inside the chain

Chain structure output

The installer enables you to view the chain structure which is part of all custom LSPs. The syntax is:

```
RegisterLSP –m
```

A sample output:

```
32-bit Winsock LSP Map:

  0 LSP:  TestLSP    DLL 'C:\WINDOWS\system32\nonifslsp.dll'
ID: 1020
      LSP Installed under 2 GUIDs
          {D7E3B4EA-D34A-4FE7-BF66-02B5E0A38BBF}
          {4EEC5B94-2F44-4D3C-9AB1-9D13658880C8}
      Layer 1021  "TestLSP over [MSAFD Tcpip [TCP/IP]]"
            Chain 2 [ 1020 1001 ]
      Layer 1022  "TestLSP over [MSAFD Tcpip [UDP/IP]]"
            Chain 2 [ 1020 1002 ]
      Dependent LSPs:
          None
```

What we see here is a single LSP installed with two chains, each composed from that LSP's dummy entry and the base provider (for further information on LSP chains, please refer to my book.)

Displaying layered LSPs only

To display only the layered LSPs without the base providers and dummy entries, use the following syntax:

```
RegisterLSP -l
```

A sample output:

```
Winsock 32-bit Catalog:
=======================
1020 - TestLSP
```

Verbose display

During installation, every installer defines a number of states that affect the LSP during runtime, (e.g., whether the LSP uses an IFS or a non-IFS handle.) Verbose display outputs a list of these predefined states. It is used mostly to debug installation problems. This option can be used in conjunction with two flags:

```
RegisterLSP -l -v
RegisterLSP -p -v
```

A sample output:

```
Winsock 32-bit Catalog:
=======================
Protocol: TestLSP
    Path: C:\WINDOWS\system32\nonifslsp.dll
           Address Family: AF_INET
                 Protocol: IPROTO_IP
              Socket Type:            Connectionless: NO
      Guaranteed Delivery: YES
         Guaranteed Order: YES
        Message Oriented: NO
            Pseudo Stream: NO
           Graceful Close: YES
           Expedited Data: YES
             Connect Data: NO
          Disconnect Data: NO
       Supports Broadcast: NO
      Supports Multipoint: NO
 Multipoint Control Plane: NON-ROOTED
    Multipoint Data Plane: NON-ROOTED
            QoS Supported: NO
     Unidirectional Sends: NO
    Unidirection Receives: NO
              IFS Handles: NO
         Partial Messages: NO
           Provider Flags: PFL_HIDDEN
```

```
              Provider     Id:     {46FA836F-6004-4D57-A267-
7092E9A6C8D8}
          Catalog Entry Id: 1020
    Number of Chain Entries: 0   {}
                   Version: 2
Max Socket Address Length: 16
Min Socket Address Length: 16
       Protocol Max Offset: 0
        Network Byte Order: BIG-ENDIAN
           Security Scheme: NONE
              Message Size: N/A (Stream Oriented)
         Provider Reserved: 0
```

Important note: this is only for the LSP entry and not for the dummy entries. The interpretation of this output is beyond the scope of this manual. For further information, please refer to (yes, you guessed it) my book.

How to build a scenario file

Scenario files are used with autopilot only; they have no effect on manual installs.

Why use a scenario file?

Scenario files are used when there is a known LSP for which the autopilot cannot detect the correct strategy, or when we choose to bypass this LSP even though it's possible to install our LSP in the same chain.
Note: bypassing an LSP when you don't have to do so is considered bad practice.

Using the scenario file

The scenario file is named RegisterLSP.ini and should be located in the same directory as the installer. The file format is simple: it is the name of the LSP followed by the installation strategy, separated by a comma. The installation strategies are:

- Bypass – install the LSP directly after the base providers and bypass the existing installed LSP
- Before – install the LSP to be first in chain, and all winsock commands will go through this LSP first
- After – install the LSP last in the chain and the closest to the base providers

Sample file structure:

```
ProductLSP,before
CompanyLSP,after
SomeLSP,bypass
```

Sanity checks and backup operations

Before installing an LSP, it is important to ensure that the stack is in perfect state; otherwise, if the computer network is already corrupt, the user might wrongly blame the last installed LSP.

Checking winsock integrity

When the stack is completely corrupted, it is impossible to initialize winsock. The installer can check for this condition and warn about it:

```
RegisterLSP -tw
```

If everything is okay, the installer will return 0 as error code. All other values denote stack corruption, and install should be halted immediately. Any installation after this kind of corruption is at your own risk.

Checking LSP chain integrity

Another type of corruption is when there is a legitimate stack but one of the LSP files is missing, resulting in lack of networking. To detect this, use:

```
RegisterLSP -tc
```

The installer will load and perform integrity tests on all the LSPs referenced by the stack. It will:

- Detect if the file exists
- Try to load the file as DLL
- Try to get the GUID from the LSP

If any of the tests fails, the installer warns about it by exiting with an error code other then zero, and as mentioned above, continuing to install at this point is at your own risk.

A sample output:

```
Testing winsock chain
Trying to load provider: TestLSP over [MSAFD Tcpip
[TCP/IP]], file name: C:\WINDOWS\system32\nonifslsp.dll,
result - Success
Trying to load provider: TestLSP over [MSAFD Tcpip
[UDP/IP]], file name: C:\WINDOWS\system32\nonifslsp.dll,
result - Success
Trying to load provider: MSAFD Tcpip [TCP/IP], file name:
C:\WINDOWS\system32\mswsock.dll, result - Success
Trying to load provider: MSAFD Tcpip [UDP/IP], file name:
C:\WINDOWS\system32\mswsock.dll, result - Success
Trying to load provider: MSAFD Tcpip [RAW/IP], file name:
C:\WINDOWS\system32\mswsock.dll, result - Success
Trying to load provider: RSVP UDP Service Provider, file
name: C:\WINDOWS\system32\rsvpsp.dll, result - Success
Trying to load provider: RSVP TCP Service Provider, file
name: C:\WINDOWS\system32\rsvpsp.dll, result - Success
Trying to load provider: MSAFD NetBIOS
[\Device\NetBT_Tcpip_{0E41ADAF-748D-4FE4-A76F-F164642EEE87}]
SEQPACKET 3, file name: C:\WINDOWS\system32\mswsock.dll,
result - Success
Trying to load provider: MSAFD NetBIOS
[\Device\NetBT_Tcpip_{0E41ADAF-748D-4FE4-A76F-F164642EEE87}]
```

```
DATAGRAM  3,  file  name:  C:\WINDOWS\system32\mswsock.dll,
result - Success
Trying     to     load     provider:     MSAFD     NetBIOS
[\Device\NetBT_Tcpip_{B8E72AD6-0B84-4188-8841-52340436256E}]
SEQPACKET  0,  file  name:  C:\WINDOWS\system32\mswsock.dll,
result - Success
Trying     to     load     provider:     MSAFD     NetBIOS
[\Device\NetBT_Tcpip_{B8E72AD6-0B84-4188-8841-52340436256E}]
DATAGRAM  0,  file  name:  C:\WINDOWS\system32\mswsock.dll,
result - Success
Trying     to     load     provider:     MSAFD     NetBIOS
[\Device\NetBT_Tcpip_{A0299C89-898F-4CF1-BCDC-1EE2550DFC55}]
SEQPACKET  1,  file  name:  C:\WINDOWS\system32\mswsock.dll,
result - Success
Trying     to     load     provider:     MSAFD     NetBIOS
[\Device\NetBT_Tcpip_{A0299C89-898F-4CF1-BCDC-1EE2550DFC55}]
DATAGRAM  1,  file  name:  C:\WINDOWS\system32\mswsock.dll,
result - Success
Trying     to     load     provider:     MSAFD     NetBIOS
[\Device\NetBT_Tcpip_{9593A3E5-5C81-4534-9160-0D9376807B68}]
SEQPACKET  2,  file  name:  C:\WINDOWS\system32\mswsock.dll,
result - Success
Trying     to     load     provider:     MSAFD     NetBIOS
[\Device\NetBT_Tcpip_{9593A3E5-5C81-4534-9160-0D9376807B68}]
DATAGRAM  2,  file  name:  C:\WINDOWS\system32\mswsock.dll,
result - Success
Trying    to    load    provider:    TestLSP,    file    name:
C:\WINDOWS\system32\nonifslsp.dll, result - Success


Exiting with error code: 0 which means everything went OK
```

Backing up the winsock stack

The installer has the ability to backup the stack using the following syntax:

```
RegisterLSP -br FileName
```

For example:

```
RegisterLSP -br c:\backup.dat
```

Restoring the winsock stack

An important point to understand is that the winsock is restored to its exact state from the time it was backed up, meaning that if after the backup operation some LSPs were uninstalled and files were removed, the restored state may be corrupted. The syntax is:

```
RegisterLSP -rr FileName
```

Restoring the stack using the file in the above example would look like this:

```
RegisterLSP –rr c:\backup.dat
```

Manual installation

Sometimes manual installation is required, usually for testing purposes. All the features of the autopilot can be used manually. This is possible as long as the proper IDs or names of the LSPs we wish to layer over are known.

The manual installation has two important flags that repeat in all manual installations: -i and –d (which denotes the LSP name and path).

Installing over other providers based on their IDs

To install an LSP over TCP and UDP, use the following syntax:

```
RegisterLSP –i –o TCPID –o UDPID –d LSPName
```

We assume that the ID for TCP is 1001 and for UDP is 1002, as seen in the printouts in this manual. Keep in mind, however, that on some computers these IDs do not match: make sure you know the proper IDs before using them in a manual install. A typical manual install would look like this:

```
RegisterLSP –i –o 1001 –o 1002 –d "c:\MyLsp.dll"
```

Installing an IFS LSP

If we install an IFS LSP, we must tell installer about it. We do so by using the flag –h. Installing an IFS LSP over TCP and UDP would look like this:

```
RegisterLSP –i –h –o 1001 –o 1002 –d "c:\MyLsp.dll"
```

Installing over other providers based on their names

As stated previously, using the ID is possible only when the person installing is using the target machine and knows the precise ID. That is why it is better to install the LSP using the layer name rather then the ID. This is done by using the –z flag instead of –o (when using IDs). Installing over TCP and UDP would look like this:

```
RegisterLSP –i –z "MSAFD Tcpip [TCP/IP]" –z "MSAFD Tcpip
[UDP/IP]" –d "c:\MyLsp.dll"
```

Installing over another LSP based on either its name or its ID

This is done exactly the same as with a base provider, the only difference being the name or ID of the LSP we wish to layer on. Suppose we have the following stack:

```
Winsock 32-bit Catalog:
=======================
1021 - TestLSP over [MSAFD Tcpip [TCP/IP]]
1022 - TestLSP over [MSAFD Tcpip [UDP/IP]]
1001 - MSAFD Tcpip [TCP/IP]
1002 - MSAFD Tcpip [UDP/IP]
1003 - MSAFD Tcpip [RAW/IP]
```

```
1020 – TestLSP
```

and we wish to layer over TestLSP. This is how we do it using an ID:

```
RegisterLSP –i –o 1021 –o 1022 –d "c:\MyLsp.dll"
```

This is how we do it using the name:

```
RegisterLSP –i –z "TestLSP over [MSAFD Tcpip [TCP/IP]]" –z
"TestLSP over [MSAFD Tcpip [UDP/IP]]" –d "c:\MyLsp.dll"
```

The only difference here from the previous examples is the different name or ID.

Installing over an existing chain

The previous example has revealed that we must know the LSP's exact name or IDs if we want to layer over it; but unless we without executing "RegisterLSP –p" we cannot know this information in advance. To overcome this problem, it is possible to instruct the installer to locate the LSP installed over TCP and UDP, and use its chain for the install process.

Installing LSP first in the chain

If we want our LSP to be first in the chain, we can use the –t flag. The following example shows how to do it layering over both TCP and UDP:

```
RegisterLSP –i –t –o 1001 –o 1002 –d "c:\MyLsp.dll"
```

This example uses the TCP's and UDP's explicit names:

```
RegisterLSP –i –t –z "MSAFD Tcpip [TCP/IP]" –z "MSAFD Tcpip
[UDP/IP]" –d "c:\MyLsp.dll"
```

Installing LSP last in the chain

We may want our LSP to be last in the chain, and closest to the base providers. For this we use the –j flag instead of the –t flag used in the previous example. This is how we do it using TCP/UDP IDs:

```
RegisterLSP –i –j –o 1001 –o 1002 –d "c:\MyLsp.dll"
```

This example uses the TCP's and UDP's explicit names:

```
RegisterLSP –i –j –z "MSAFD Tcpip [TCP/IP]" –z "MSAFD Tcpip
[UDP/IP]" –d "c:\MyLsp.dll"
```

64Bit

An LSP compiled using 32bit can work in 32bit and 64bit environments, but an LSP compiled using 64bit can only work in a 64bit environment. Furthermore, a 32bit installer can install only 32bit LSPs, but a 64bit installer (which can operate only under a 64bit environment) can install both 32bit and 64bit LSPs.

Working with the DLL installer

When is it better to use the DLL

- Under Vista, the DLL has the privileges of its spawning process, which means that if the caller is running with UAC elevated mode, there is no need to get another elevation for the installer.
- The installer has the ability to retrieve the list of installed LSPs inside an array, so it can be further inspected. (This can also be done with the normal installer; however, it requires further parsing by the user.)

Linking with the DLL

This can be done either by linking with the static library or using the *LoadModule* API command. The installer comes with a sample that shows how to link with the static library.

Installer API

Working with the DLL installer is almost the same as working with the command line installer. Programmatically, it accepts the same parameters and performs the same as the command line installer.

AddParam, Execute, ResetParam

AddParam is used to add a single parameter to the installer; Execute is used to actually perform the command. ResetParam is used if there is no need to perform another call to the installer, and removes all the parameters added by previous call to AddParam. The following sample runs an autopilot install of an LSP, and then removes all installed LSPs:

```
//The following code is equivalent to: RegisterLSP –b –d c:\nonifslsp.dll
      AddParam("-b");
      AddParam("-d");
      AddParam("c:\\nonifslsp.dll");
      Execute();

//Need to delete the previous parameters
ResetParam();

//Now perform the remove
AddParam("-f");
Execute();
```

Subverting output

The installer in DLL mode doesn't output to the screen like the command line installer, so there are functions that send the output into a buffer.

GetOutputDataSize

Returns the number of bytes required to get the output buffer.

GetOutputData

Gets the actual data from the buffer. If successful, the return value is the buffer size. If the buffer is too small, the return value will be –1. After successful retrieval of the data, the output is erased. A sample usage:

```
//First get the size needed
int iSize=GetOutputDataSize();

//Do we have data?
If (iSize)
{
     //Allocate a buffer to accommodate
     char* pData;
     pData=new char[iSize];

     //And get the data
     if (GetOutputData(pData,
                      iSize)>0)
          //Show it, or do some other stuff
          printf("%s\n",pData);

     //Cleanup
     delete [] pData;
}
```

Vista API

There are two methods that help working with Vista.

IsVista

Returns true if the OS is Vista.

IsElevated

Returns true if UAC is elevated. If the OS is not Vista, the return value will always be true. All install/uninstall operations on Vista require UAC elevation, without which the install/uninstall will fail.

Sample code:

```
//Are we inside Vista?
if (IsVista())
     //Are we elevated?
     if (!IsElevated())
          //Do something about it
          printf("Vista UAC not elevated!");
```

Getting the installed LSP list

It is possible to perform a –p command and parse the output. However, there is an easier method for retrieving the data already parsed.

GetInstalledLSPs

Expects a pointer large enough in size to contain all the LSPs inside the stack. The data is sorted by LSP ID.

Sample code:

```
//Allocate big enough variable, never seen a system with 100
LSPs!
char* pData;
pData=new char[100*sizeof(InstalledLSP)];

//Perform the call
int iLSPs=GetInstalledLSPs(pData,
                               100*sizeof(InstalledLSP));

//Did we get it?
if (iLSPs>0)
     //Do something, e.g., print it
     for (int iCounter=0;
           iCounter<iLSPs/sizeof(InstalledLSP);
           iCounter++)
          printf("%i    -    %s\n",pData[iCounter].    dwID,
pData[iCounter]. aName);

//Cleanup
delete [] pData;
```

Working with the COM installer

The COM installer is a COM DLL that can be used from every COM-capable application: VC, VB, .Net, etc.

It exposes the same API functionalities as does the DLL installer. The only difference is with the parameters and return value type, which are variant compliant to adhere to the COM standards.

Registering the DLL

Since the DLL is COM based, it must be registered as such. To do so, run the following command inside the DLL's directory:

```
Regsvr32 RegisterLSPCOM.dll
```