

Written by : Barak Weichselbaum, all rights reserved 2000-2008

Site: <http://www.komodiam.com/>

Email : [barak@komodiam.com](mailto:barak@komodiam.com)

## LSP management

This article requires the reader to have knowledge in winsock2 technology; I personally think that without such knowledge one cannot learn LSP.

### **1. What is LSP?**

LSP – Layered Service Provider, is a component that intercepts winsock2 calls, has the ability to manipulate them, and then passing them to the winsock2 provider (optionally).

There are two kinds of LSP frameworks:

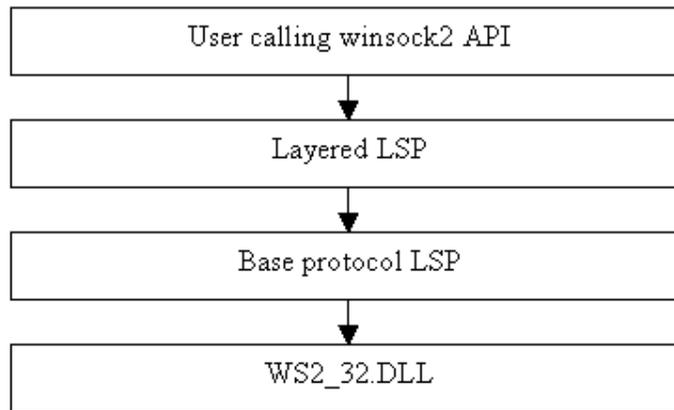
- Transport service provider.
- Name space provider.

### **2. LSP – Transport service provider.**

LSP is organized in a chain structure, i.e. the first call goes to the first LSP in the chain, the LSP processes the call, and then call the next LSP in the chain, the last LSP calls the actual winsock2 functions.

Each “service” has a different LSP chain (i.e. LSP for TCP,UDP,raw sockets, netbios), these LSPs are called base protocol because they call the actual winsock2 functions.

The LSPs in the chain which aren't base protocol are called layered provider. (They implement only higher-level calls)



### **3. LSP – Name space provider (NSP)**

This article will focus on transport provider LSP and not of Name space provider.

Name space provider is a way to allow protocol to handle its naming methods, i.e. for TCP/IP the name space provider will query the DNS and resolve it's IP address.

It's mostly used to extend domains on the web, for example, you can sell domains the end with ".lsp", of course they are not official however if the resolving computer has NSP installed on you would be able to resolve the ".lsp" extension (I'm assuming that the NSP is made by the seller of the ".lsp" domain and act to resolve it)

## **4. LSP – where to start.**

Well, documentation on LSP is scarce, and the MSDN documentation will lead you to nowhere. (Or at least let you waste valuable time with experiments).

Microsoft's default LSP samples have matured and are quite good, however without the proper guidance they can look scary!

Part of this document is my LSP template. (I took Microsoft's LSP, and wrapped it in VC6 project, I just hate makefiles)

Writing LSP from scratch is time consuming and basically and is not something most programmers will do, I actually thought about writing such LSP that will be pure C++ with many good features and ideas however it's time consuming and I focus on more areas of programming.

## **5. LSP – Let's get technical – and a warning.**

LSP is a DLL, and to get it working you must register it. (Part of the sample is the registering software – “LSP installer”, again taken from Microsoft's sample).

I strongly advice backing up the registry entry that is being modified, because later on, removing the LSP will be easy and will not hamper the OS normal operation. (Messing with LSPs without knowing why and what can destroy the OS networking ability to work normally, or even crash at boot time, so I strongly recommend on the backing up).

The LSP registry is :  
HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\WinSock2

Back it up, and all the sub registry keys.

## 6. LSP - loading sequence.

Because LSP is a DLL our first stop will be at the DllMain, the LSP initializes only its basic variables. (Be careful when adding code to the LSP, In the DllMain you must do minimum initializations, otherwise it can lead to unexpected results. I once tried to spawn threads, or to create events on the DllMain, and I spend the night understanding what went wrong, special notice goes for CoInitialize and CoInitializeEx which the documentations states that they shouldn't be called from the DllMain).

When the DLL is unloaded the LSP will deallocate all of its resources.

The LSP is loaded in each process that has winsock2 calls, that are using the base protocol the LSP has layered (so you may have many running instances of your LSP).

The second important LSP function is : WSPStartup (which is exported at the LSP's .def file)

This function is called when the user calls WSASStartup.

Basically the LSP loads all the layered protocol beneath him in the chain, and initializes its calling tables. (Dynamic table for calling its WSP functions).

## 7. Installing the LSP.

Every LSP has a unique GUID, make sure you change the defaults GUID. (It's on Provider.CPP, variable ProviderGuid, you can use GuidGen, which is supplied with the Visual Studio to generate a new GUID).

The latest MS "LSP Installer" sample can take an LSP dynamically and get the GUID from an exposed method called "GetLspGuid".

This guide shows how to use the default installer, however in order to have a 100% successful install you need to write extra 10,000 line of code. The good news, we already did it, you can check out our "Komodia's advanced LSP installer" webpage at:

<http://www.komodia.com/index.php?page=rlsp.html>

## 8. Running the registrant.

The registrant (RegisterLSP), is command line based application.

Lets start with the most useful parameters: RegisterLSP -p (It will list all the LSPs configured on the machine), a sample list will look like this:

```
- 1001MSAFD Tcpip [TCP/IP[
- 1002MSAFD Tcpip [UDP/IP[
- 1003MSAFD Tcpip [RAW/IP[
- 1004RSVP UDP Service Provider
- 1005RSVP TCP Service Provider
- 1070MSAFD NetBIOS [\Device\NetBT_Tcpip_{F4754F49-E15B-43DC-B8D3-
1BA4BB08F224{
 [SEQPACKET 0
- 1071MSAFD NetBIOS [\Device\NetBT_Tcpip_{F4754F49-E15B-43DC-B8D3-
1BA4BB08F224{
 [DATAGRAM 0
- 1072MSAFD NetBIOS [\Device\NetBT_Tcpip_{0EC4D370-C932-4F6A-BE69-
7EE8A6653B7C{
 [SEQPACKET 1
- 1073MSAFD NetBIOS [\Device\NetBT_Tcpip_{0EC4D370-C932-4F6A-BE69-
7EE8A6653B7C{
 [DATAGRAM 1
```

The number on the left is the LSP ID, and the string on the right is the LSP's description.

Now that we have the list we can layer our LSP on top of the base protocols. (Base protocols for TCP on this case is 1001, UDP 1002, and raw sockets is 1003, however it may change from machine to machine, because on most laptops there is a IrDA (Infra Red), LSP installed which will be 1001, TCP will be 1002)

Usually for most applications we will want to install our self on top of TCP, this is done like this: RegisterLSP -i -o 1001 -d LSPFilename

-i, is for installing our LSP.

-o, is the ID of the base protocol we are layering on top. (if we wanted to layer more then one layer we need to specify the -o as much as we need, i.e. RegisterLSP -i -o 1001 -o 1002 -d LSPFilename)

-d, is for the LSP file location

Now after we installed our LSP, we can view it with: RegisterLSP -p, or RegisterLSP -l, which displays layered protocol only.

If you want to install your LSP over all the protocols it is done like this: RegisterLSP -a -d LSPFilename

About removing LSPs, I would just restore the backup registry we made, but it can be done like this: RegisterLSP -r

The option I would not use is : RegisterLSP -f, which removes all layered entries. (Can delete some useful LSPs).

## 9. Install issues

Internet explorer (all versions), has a nice “feature” (remember it’s a feature not a bug), it uses UDP for keepalives, and it sends UDP sockets into the TCP functions (or vice versa), anyway it’s not a legal behavior, but we are to blame (NOT!), to fix this, make sure whenever you layer TCP, layer over UDP as well. (this will solve the problem)

## 10. Modifying the LSP

OK we have a compiling and working LSP, now we want to add some features to it.

Every Winsock call that is prefixed by WSA, has a WSP call in the LSP. WSARecv is mapped to WSPRecv, so if we want to log all data that is entering the system we could:

```
int WSPAPI WSPRecv(  
    SOCKET          s,  
    LPWSABUF       lpBuffers,  
    DWORD          dwBufferCount,  
    LPDWORD        lpNumberOfBytesRecvd,  
    LPDWORD        lpFlags,  
    LPWSAOVERLAPPED lpOverlapped,  
    LPWSAOVERLAPPED_COMPLETION_ROUTINE  
    lpCompletionRoutine,  
    LPWSATHREADID  lpThreadId,  
    LPINT          lpErrno)
```

This is the function, now in the beginning (of all WSP functions that has a socket ID as a parameter), the LSP is looking for the socket context. (an internal structure to keep information about each socket, if you want to keep information on each socket, this is the place to put it in, will be covered later)

After the socket context is found, the LSP checks if the operation is overlapped.

If it is, saves the overlap data inside the overlap structure in the context and queues it for the function that handles the overlapped operation. (OverlappedManagerThread is the actual overlap dispatcher and IntermediateCompletionRoutine does the work)

If not the LSP calls the next LSP on the chain.

So lets suppose you wanted to get the data that the buffer will end up with. (looking for a specific string, if you wanted to implement context checking for an IDS)

You will have to intercept to places, first one is in the IntermediateCompletionRoutine, incase the receive call was overlapped, and the seconds is after the call to the next LSP (if it wasn't overlapped).

Don't forget to check the error flag for errors, if there was an error the buffer has some random data.

## 11. Blocking incoming connections

Most people want to create LSP for this reason.

To do it, you need to modify WSPAccept:

```
SOCKET WSPAPI WSPAccept(  
    SOCKET          s,  
    struct sockaddr FAR * addr,  
    LPINT           addrlen,  
    LPCONDITIONPROC lpfnCondition,  
    DWORD_PTR      dwCallbackData,  
    LPINT           lpErrno)
```

You can get the address that is trying to connect inspecting the variable addr.

If you decided to not allow this connection just return the value INVALID\_SOCKET.

That's it, easy ah.

## 12. Keeping additional socket data.

To keep your socket data, you will have to expand the context structure. It is defined at Provider.h and is called SOCK\_INFO.

Let's suppose we want to add a running number to each socket, we will add an int variable, and the structure will look like this:

```
typedef struct _SOCK_INFO
{
    SOCKET ProviderSocket;           // lower provider socket
    handle
    SOCKET LayeredSocket;           // app's socket handle
    DWORD dwOutstandingAsync;       // count of outstanding
    async operations
    BOOL bClosing;                  // has the app closed the
    socket?
    DWORD BytesSent;                 // Byte counts
    DWORD BytesRecv;
    HANDLE hIocp;                    // associated with an
    IOCP?

    HWND hWnd;                       // Window (if any)
    associated with socket
    UINT uMsg;                        // Message for socket
    events

    CRITICAL_SECTION SockCritSec;

    struct _PROVIDER *Provider;
    struct _SOCK_INFO *prev,
    * next;
    int iRunningNumber;
} SOCK_INFO;
```

The running number variable is iRunningNumber.

A new socket context is created at: CreateSockInfo:

```
SOCK_INFO *CreateSockInfo(PROVIDER *Provider, SOCKET
ProviderSocket, SOCK_INFO *Inherit)
```

A variable to look for is Inherit, it specifies if the data is taken from another SOCK\_INFO structure (like when a socket is accepted).

After all the copies and initialization we will add:

```
//Original code
    NewInfo->ProviderSocket      = ProviderSocket;
    NewInfo->bClosing            = FALSE;
    NewInfo->dwOutstandingAsync  = 0;
    NewInfo->BytesRecv          = 0;
    NewInfo->BytesSent          = 0;
    NewInfo->Provider            = Provider;
    NewInfo->hWnd                = (Inherit ? Inherit-
>hWnd : 0; (
    NewInfo->uMsg                = (Inherit ? Inherit-
>uMsg : 0; (

//Our new code

    static iID=1;
    NewInfo->iRunningNumber=iID++;
```

And in each call the context info will contain this data, for our use.

### **13. Communicating with the LSP.**

Since LSP is a DLL not running in our process, we will have to use IPC to talk to it (Inter Process Communication).

I created an implementation of sockets using shared memory, and I used it to talk to my LSP.

But there are many ways to do it, mail slots, COM, and pipes.

Every one will have its own preference, but I will not elaborate since this is not the objective of this article.

### **14. Now what.**

Well now that you understand LSP (I hope), the MSDN will not look so complicated now, and you can spend your valuable time on creating the functionality you need (and not on figuring out what went wrong, it took me two month until I got the LSP up and working, without any crashes).

## 15. Komodia LSP skeletons

With this guide you have two samples: one is the original from MS, and the other one is the same files just compiled but we at Komodia aren't using the exact files for our projects, we have modified the LSP installer because: MS LSP installer is quite deprecated, it may be useful to use on test environments but when used on live machines with existing LSP installed it require a lot of work, we added around 10,000 lines of code to handle all sorts exceptions and scenarios to continue where MS stopped (I wrote about it in section 7, you can also view our installer web page at: <http://www.komodiam.com/index.php?page=rlsp.html>)