Kayaker's - Guide to creating a Softice Kernel Debugger Extension (KDExtension)

I had always planned on providing a skeleton project for developing Softice extensions, both in MASM and C++, in the hopes of allowing people to develop and share their own Softice extension plugins.
Perhaps I can outline a brief recipe here. Softice extension drivers are based on the same WINDBG_EXTENSION_APIS interface used by WinDbg. One critical difference from a regular driver
is how DriverEntry is handled.

Normally ntoskrnl creates a mostly blank DRIVER_OBJECT structure for the driver being loaded, then calls its DriverEntry routine as an indirect call. You can trace back from a regular DriverEntry routine and find where this is done, it's a fixed address in ntoskrnl that is used for all normal drivers. The prototype for a regular DriverEntry is:

DriverEntry(PDRIVER_OBJECT pDriverObject, PUNICODE_STRING RegistryPath)

With a Softice extension driver the stack parameters are different on driver entry. As a result of the dependancy to NTICE you declared on installation in the CreateService call, ntoskrnl passes control
to Softice for final loading of the driver. The prototype on DriverEntry is more like a regular DLL main entry routine:

DriverEntry(long pBaseAddress, long fdwReason, long reserved)

If you trace back from a Sice extension DriverEntry you will land in ntice.sys code rather than ntoskrnl and find that it was called like this:
Code:

```
; 6A 00 push 0 ; 6A 01 push 1
; DLL_PROCESS_ATTACH
; FF 76 17 push dword ptr [esi+17h]
; Base Address (MZ header)
; FF 56 1B call dword ptr [esi+1Bh]
; DriverEntry_fdwReason_LoadKDExtension Also, DriverEntry is called on unloading as well
; 6A 00 push 0
; 6A 00 push 0
; DLL_PROCESS_DETACH
; FF 76 17 push dword ptr [esi+17h]
; Base Address (MZ header)
; FF 56 1B call dword ptr [esi+1Bh]
; DriverEntry_fdwReason_UnloadKDExtension
```

What is important here is that you have no direct reference to PDRIVER_OBJECT. So you can't even use IoCreateDevice or IoCreateSymbolicLink without first finding a pointer to the Driver Object. A pointer to
DRIVER_OBJECT does sit on the stack but varies with Softice versions

```
; [ebp+24h] ; pDRIVER_OBJECT for DS3.1
; [ebp+28h] ; pDRIVER_OBJECT for DS2.7
```

What I do is to find an absolute pointer to DRIVER_OBJECT using ObReferenceObjectByName, then just proceed as with a normal DriverEntry routine.

Now, what I've described is a method for a "normal" Softice KDExtension. i.e. install/register the driver with a dependancy on NTICE, incorporate a WinDbgExtensionDllInit interface routine, create individual extension command modules, and use some form of modified DriverEntry to allow for the parameter differences. None of this is documented per se, but is the method I use.

Sten does things a little different. While he's changed his method a bit over time, I see he's now able to avoid the problem of the missing PDRIVER_OBJECT entirely. In recent versions IceExt was registered
as a KDExtension (KDExt) by directly modifying Softice code rather than creating an entry in the Registry. If you follow the Softice code I posted above (continue from DriverEntry_fdwReason_LoadKDExtension),
you see where Sice then parses the EXPORT_DIRECTORY of your KDExt and copies it into some global buffer. I *believe* that IceExt modifies this directly, "fooling" Sice into having already registered this KDExt,
please correct me if I'm wrong.

In addition, in tracing the loading of IceExt v.67 I now see that the entire sequence of Softice having to do the final loading of the KDExt driver has been eliminated. It's loaded as a normal driver from the usual ntoskrnl
path. This must be the use of TARGETTYPE MINIPORT you mentioned. I hadn't seen that yet, IceExt is being loaded as a miniport driver it seems. One benefit of doing it this way is being able to change the name of the
installed driver and hide the fact that IceExt even exists. Nice trick Sten

Back to the rest. No, you only require a WinDbgExtensionDllInit routine. ExtensionApiVersion or CheckVersion are not used by Softice (I don't think WinDbg even uses the latter).
.....

Here is a rough guide to creating a Softice Extension.
Have in place a standard skeleton driver + driver install routine.

The critical parts of the install routine:
Code:

```
aNTICE db "NTICE",0,0
; double null terminated
; Create the service with a dependancy on NTICE invoke
    CreateService, hScManager, offset ServiceName, offset ServiceName,\
SERVICE_ALL_ACCESS, SERVICE_KERNEL_DRIVER,\ SERVICE_DEMAND_START,
SERVICE_ERROR_NORMAL,\ ADDR RootFilePath, 0,0,offset aNTICE,0,0
    Use regular registry writing routines to register the name of your driver under
HKLM\SYSTEM\CurrentControlSet\Services\NTICE\KDExt ensions KDExtensionName db
"KDExtname.sys;",0 note the ";"
 at the end, concatenate strings for multiple extension drivers
```

A DriverEntry routine modified to get a pointer to its own Driver Object:
Code:

```
//////////////////////////////////////////////////////////////////////
//                              External Declarations
//////////////////////////////////////////////////////////////////////

EXTERN_C PVOID IoDriverObjectType;

EXTERN_C
NTSYSAPI
NTSTATUS NTAPI ObReferenceObjectByName(
        IN PUNICODE_STRING ObjectPath,
        IN ULONG Attributes,
        IN PACCESS_STATE PassedAccessState OPTIONAL,
        IN ACCESS_MASK DesiredAccess OPTIONAL,
        IN POBJECT_TYPE ObjectType,
        IN KPROCESSOR_MODE AccessMode,
        IN OUT PVOID ParseContext OPTIONAL,
        OUT PVOID *ObjectPtr
);


//////////////////////////////////////////////////////////////////////
//                                      GetDriverObject
//
// Returns DriverObject pointer
//////////////////////////////////////////////////////////////////////

NTSTATUS GetDriverObject(PWSTR pwszDriverName)
{

NTSTATUS status;
UNICODE_STRING DeviceName;
PDRIVER_OBJECT pDriverObject = 0;

        RtlInitUnicodeString(&DeviceName, pwszDriverName);

        status = ObReferenceObjectByName(&DeviceName, OBJ_CASE_INSENSITIVE,
                NULL, 0, (POBJECT_TYPE)IoDriverObjectType,
                KernelMode, NULL, (PVOID*)&pDriverObject);

        if (!pDriverObject) {
                return 0;
        }

        ObDereferenceObject(pDriverObject);


        return (long) pDriverObject;

}       // end GetDriverObject

//////////////////////////////////////////////////////////////////////
```

The required WinDbgExtensionDllInit routine.
See WinDbg docs for details:

Code:

```
WINDBG_EXTENSION_APIS ExtensionApis
 //==========================================
 // WinDbgExtensionDllInit                                    //
==========================================
// WinDbg (or Softice) calls this function when the user loads the
// extension DLL. Its job is to save the address of the callback table
// so that other parts of the DLL can use it. This function is required.
// // lpExtensionApis is a pointer to the WINDBG_EXTENSION_APIS structure,
// a list of internal functions that can be called from WinDbg or Softice
// extensions.
//==========================================

VOID WinDbgExtensionDllInit( PWINDBG_EXTENSION_APIS lpExtensionApis, ULONG
MajorVersion, ULONG MinorVersion) {
   // wdbgexts.h expects variable name of "ExtensionApis"
   // to store the address of WINDBG_EXTENSION_APIS
 ExtensionApis = *lpExtensionApis;

 //=============================================

return; } // end WinDbgExtensionDllInit
```

And finally, a basic KDExtension available as a ! command in Softice:
They run at the same elevated IRQL level as Softice - do not use INT1/INT3 to debug!
Code:

```
/////////////////////////////////////////////////////////////
//  KDExtension simply to echo user input
/////////////////////////////////////////////////////////////

#include <windef.h>
#include "wdbgexts.h"


DECLARE_API ( _echome )
{

        dprintf ("%s\n", args);
        return;
}
```

I hope this helps as a short guide to creating KDExtensions. Who says Softice is useless?

Regards,
Kayaker