© The Assembly-Programming-Journal, Vol.2, No.1. (2004)
http://www.Assembly-Journal.com

Kernel Mode Driver Tutorial for MASM32 Programmers Part 2 – Services

Author: Four-F

Abstract

Source code: KmdKit\examples\simple\Beeper

You may wonder how user-mode services related to the kernel-mode drivers. Actually they are completely different animals altogether. But before we can communicate with the device driver we have to install and start it at first. So, let's conform the interface rules regarding interaction with the services.

Contents

2.1 Services

Windows NT has a mechanism to start processes that provide services not tied to an interactive user. Such processes are called ***services***. A good example of a service might be a Web server. Most of the services don't have any user interface. It's a sole category of applications working that way. Services can be started at the system startup time or they can be also started manually. In that sense the device drivers are very similar to the services.

Windows NT also supports a ***driver service***, which conforms to the device driver protocols for Windows NT. It's similar to the user-mode service. So, service can be referred either to a user-mode server process or to a kernel-mode device driver. Microsoft had for unknown reasons mixed up user-mode services and kernel-mode drivers. Therefore further narration can seem a little bit confusing, since I will use at times term "driver", at the other times - "service". But this article deals with kernel-mode device drivers only. And you should always consider it like a "driver". I will explicitly note when necessary to separate "service" from "driver". Also keep in mind that the documentation describing the functions to manipulate with the services is rather ambiguous at times. Many functions discussed in this section apply to both services and device drivers, but I will emphasize on device drivers and omit discussing services.

There are three types of components involved in making Windows NT services work:

- ***Service Control Manager*** (SCM). The SCM is responsible for starting the service, communicating with it and so on.
- ***Service Control Program*** (SCP). The SCP communicates with the SCM telling it when to start or stop service and so on.
- A ***service program*** that contains executable code. And as I noted earlier the service is considered as the kernel-mode device driver.

As I have already said, we'll study the driver itself in the next part, and now we'll concentrate on the first two components.

2.2 The Service Control Manager

The SCM lives in \%SystemRoot%\System32\Services.exe. Winlogon process starts the SCM early during the system boot. It then scans the contents of the registry under the key HKLM\SYSTEM\CurrentControlSet\Services, creating an entry in the service database for each key it encounters. A database entry includes all the service-related parameters defined for a service. If service or a driver is marked for auto-start the SCM starts it and detects startup failures.

To gain some insight about it, start the Registry Editor (\%SystemRoot%\regedit.exe), then open HKLM\SYSTEM\CurrentControlSet\Services\ and explore its content.

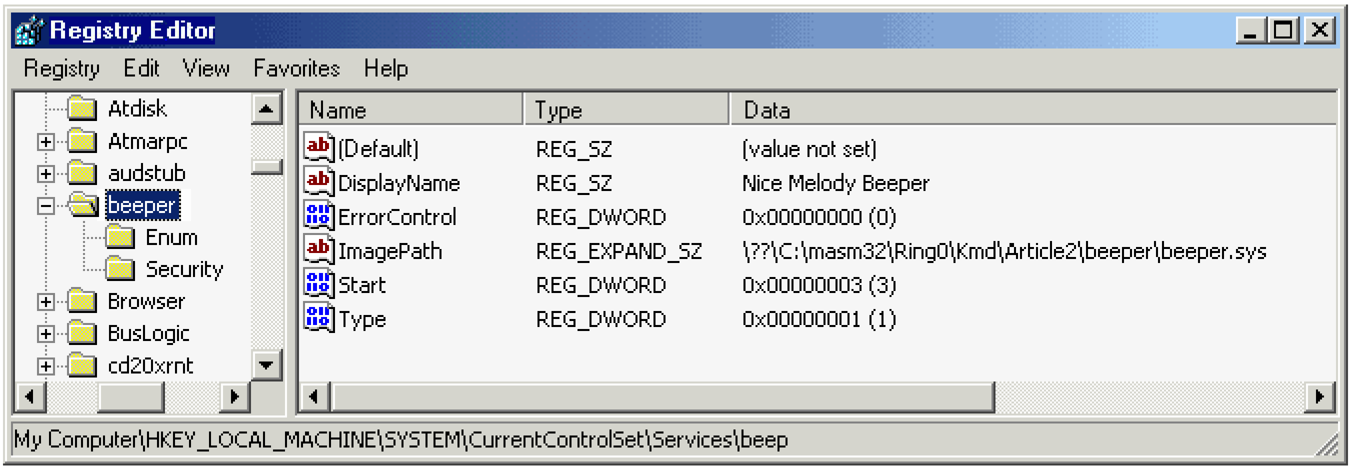
To view the list of installed services (not drivers), select Administrative Tools from Control Panel, and then select Services.

Running Computer Management you can list the installed drivers. (From the Start menu, select Programs, Administrative Tools, and then Computer Management; or from Control Panel, open Administrative Tools and select Computer Management.) From within Computer Management expand System Information and then Software Environment, and open Drivers (Unfortunately, this feature is unavailable since Windows XP).

Having analyzed the content of these three windows, you will notice that they coincide in many respects.

The HKLM\SYSTEM\CurrentControlSet\Services\ contains a subkeys, denoted by an internal name of the driver or service. Each subkey includes all the service-related parameters.

Let's consider a minimum possible set of parameters necessary to install device driver. As an example, we'll take the beeper.sys driver (we'll talk about the driver itself next time).



Name	Type	Data
(Default)	REG_SZ	(value not set)
DisplayName	REG_SZ	Nice Melody Beeper
ErrorControl	REG_DWORD	0x00000000 (0)
ImagePath	REG_EXPAND_SZ	\??\C:\masm32\Ring0\Kmd\Article2\beeper\beeper.sys
Start	REG_DWORD	0x00000003 (3)
Type	REG_DWORD	0x00000001 (1)

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\beep

Figure 2-1. Registry key for beeper.sys driver

Parameter	Description	
DisplayName	- Name of service to be used by user interface programs. If no name is specified, the name of the service's registry key becomes its name.	
ErrorControl	- If a driver reports an error in response to the SCM's startup command, this value specifies the level of error control and determines SCM's reactions. Two values can be of interest for us:	
	SERVICE_ERROR_IGNORE (0)	- The I/O Manager ignores errors the driver returns but continues the startup operation. Nothing is logged;
	SERVICE_ERROR_NORMAL (1)	- If the driver fails to load or initialize, startup should proceed with a warning display to the user. And an event to the System Event Log is written.
	You can view an event description by selecting Administrative Tools > Event Viewer and double-clicking on an Event Log entry. For example, the beeper driver does all useful job at initialization stage (in the DriverEntry routine), then it returns an error code to be removed from the memory since it can't do anything more. The ErrorControl parameter for beeper driver is equal to SERVICE_ERROR_IGNORE, so no logging occured.	
ImagePath	- Specifies the fully qualified path of the driver's image file. If ImagePath is not specified, the I/O Manager looks for drivers in \%SystemRoot%\Drivers directory.	
Start	- Specifies when to start the driver. There can be useful only two values to us:	
	SERVICE_AUTO_START (2)	- A driver is started during system

		startup.
	SERVICE_DEMAND_START (3)	- A driver is started by the Service Control Manager in response to an explicit user demand.
	If driver has Start specified as SERVICE_AUTO_START (2) it will be started by the SCM during system startup. Such drivers are called auto-start services. If the driver depends on any other drivers SCM will starts those drivers too (To control the order of loading device drivers use the Group, Tag and DependOnGroup values and services use Group and DependOnService). There are also other flags indicating auto-start, for example, SERVICE_BOOT_START (0). Only device drivers can specify it. The I/O Manager loads such drivers before any user-mode processes execute, and therefore before the SCM starts.	
Type	- Specifies the type of service. Since we are going to deal with device driver the only value we can use is SERVICE_KERNEL_DRIVER (1).	

Having looked on figure 2-1 what can we tell about beeper.sys driver? Well, Kernel-mode driver beeper is resides in C:\masm32\Ring0\Kmd\Article2\beeper directory. It has display name "Nice Melody Beeper", started on demand, possible errors are ignored and not logged.

What prefix "\??" in the path to the driver's image file means you will know later.

If we want to start the driver not presented in the SCM database, it can be done dynamically, at any moment, with the help of the service control program (device control program to be more precise, but there in no such concept in Microsoft terminology).

2.3 The Service Control Program

As follows from its name, the service control program is intended to control the service or device driver. It does this under the SCM supervision, calling the appropriate functions. All of them are exported by the module \%SystemRoot%\System32\advapi.dll (Advanced API).

Here is the code of SCP, which will control the beeper.sys driver.

```
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::
;
;  Service Control Program for beeper driver
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::

.386
.model flat, stdcall
option casemap:none

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::
;                             I N C L U D E   F I L E S
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::

include \masm32\include\windows.inc

include \masm32\include\kernel32.inc
include \masm32\include\user32.inc
include \masm32\include\advapi32.inc

includelib \masm32\lib\kernel32.lib
includelib \masm32\lib\user32.lib
includelib \masm32\lib\advapi32.lib

include \masm32\Macros\Strings.mac

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::
;                                       C O D E
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:::::::::::::::::::::::::::::::::

.code

start proc

local hSCManager:HANDLE
local hService:HANDLE
local acDriverPath[MAX_PATH]:CHAR

    invoke OpenSCManager, NULL, NULL, SC_MANAGER_CREATE_SERVICE
    .if eax != NULL
        mov hSCManager, eax

        push eax
        invoke GetFullPathName, $CTA0("beeper.sys"), sizeof
acDriverPath, addr acDriverPath, esp
        pop eax

        invoke CreateService, hSCManager, $CTA0("beeper"),
$CTA0("Nice Melody Beeper"), \
                SERVICE_START + DELETE, SERVICE_KERNEL_DRIVER,
SERVICE_DEMAND_START, \
                SERVICE_ERROR_IGNORE, addr acDriverPath, NULL,
NULL, NULL, NULL, NULL
        .if eax != NULL
            mov hService, eax
            invoke StartService, hService, 0, NULL
            invoke DeleteService, hService
            invoke CloseServiceHandle, hService
        .else
            invoke MessageBox, NULL, $CTA0("Can't register
driver."), NULL, MB_ICONSTOP
        .endif
        invoke CloseServiceHandle, hSCManager
    .else
        invoke MessageBox, NULL, $CTA0("Can't connect to Service
Control Manager."), \
                            NULL, MB_ICONSTOP
    .endif

    invoke ExitProcess, 0

start endp

;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::
;
;::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::::::::::::::::::::

end start
```

2.3.1 Establishing a connection to the SCM

The first thing we have to do is to call OpenSCManager function to establish a connection to the SCM on the specified computer and to open the specified database.

```
OpenSCManager proto lpMachineName:LPSTR, lpDatabaseName:LPSTR,
dwDesiredAccess:DWORD
```

Parameter	Îďčńŕíčĺ
lpMachineName	- Points to a null-terminated string that names the target computer. If the pointer is NULL or if it points to an empty string, the function connects to the SCM on the local computer.

lpDatabaseName	- Points to a null-terminated string that names the SCM database to open. This string should specify ServicesActive. If it is or NULL, the ServicesActive database is opened by default.
	`.const`
	`szActiveDatabase db "ServicesActive", 0`

```
SERVICES_ACTIVE_DATABASE equ offset szActiveDatabase
```

Since we are not going to open any other SCM database, except for the active one, we simply specify NULL.

dwDesiredAccess	- Specifies the access right to the SCM. This parameter tells the SCM what we intend to do with its database. Three values can be useful to us:	
	SC_MANAGER_CONNECT	- Enables connecting to the SCM. This access type is implicitly specified by default (if you simply pass 0). Very strange, but the documentation tells nothing about what particularly we can do having this access type. But many actions can be done. We can start and stop the driver, and even delete its entry from the SCM database;
	SC_MANAGER_CREATE_SERVICE	- Enables calling of the CreateService function to create a service object and add it to the database. Actually having this access type creating a service is not a sole thing we can do. Since the SC_MANAGER_CONNECT flag is set by default, we can do all possible with this access type. Though it's not obvious too;
	SC_MANAGER_ALL_ACCESS	- Gives full access to the SCM database.

We establish a connection to the SCM in this way:

```
    invoke OpenSCManager, NULL, NULL, SC_MANAGER_CREATE_SERVICE
    .if eax != NULL
        mov hSCManager, eax
```

If the OpenSCManager succeeds, the return value is a handle to the specified SCM database. We'll pass it to other functions to manipulate the SCM database.

By the way, I've forgotten to say that the installation of kernel-mode device driver requires an account with administrator privileges. It provides the necessary security. So normal users cannot add and execute privileged code without the proper authority. Therefore, it's assumed here that you have appropriate privilege level.

2.3.2 Installing new driver

Once the SCM has been opened, we add our driver to its database by the call to CreateService. Here is its prototype. CreateService has thirteen parameters. But don't panic. Actually everything is rather simple.

```
CreateService proto hSCManager:HANDLE, lpServiceName:LPSTR,
lpDisplayName:LPSTR, \
                    dwDesiredAccess:DWORD, dwServiceType:DWORD,
dwStartType:DWORD, \
                    dwErrorControl:DWORD, lpBinaryPathName:LPSTR,
lpLoadOrderGroup:LPSTR, \
                    lpdwTagId:LPDWORD, lpDependencies:LPSTR,
lpServiceStartName:LPSTR, \
                    lpPassword:LPSTR
```

Parameter	Îďčńříčĺ	
hSCManager	- Handle to SCM database.	
lpServiceName	- Points to a null-terminated string that names the service to install. The maximum string length is 256 characters. Forward-slash (/) and back-slash (\) are invalid service name characters. This string corresponds to a name of a service registry subkey.	
lpDisplayName	- Points to a null-terminated string that is to be used by user interface programs to identify the service. This string has a maximum length of 256 characters. Corresponds to the DisplayName value under service registry subkey.	
dwDesiredAccess	- Specifies the access to the service. There can be useful following values for us:	
	SERVICE_ALL_ACCESS	- Full access to the service;
	SERVICE_START	- Enables calling of the StartService

		function to start the service;
	SERVICE_STOP	- Enables calling of the ControlService function to stop the service;
	DELETE	- Enables calling of the DeleteService function to delete the service;
	We need to do only two things: to start the driver and to remove it from the SCM database. So, we pass SERVICE_START and DELETE in this parameter. We don't have to stop the started driver since its initialization will fail.	
dwServiceType	- Specifies the type of service. We use only SERVICE_KERNEL_DRIVER. Corresponds to the Type value under service registry subkey.	
dwStartType	- Specifies when to start the service. If we want to start the driver by ourselves we pass SERVICE_DEMAND_START. If the driver should be started right after system boots, just before logon prompt appears, pass SERVICE_AUTO_START. Corresponds to the Start value under service registry subkey.	
dwErrorControl	- Specifies the severity of the error if the driver fails to start during startup. We use SERVICE_ERROR_IGNORE to ignore errors or SERVICE_ERROR_NORMAL to log possible errors. Corresponds to the ErrorControl value under service registry subkey.	
lpBinaryPathName	- Points to a null-terminated string that contains the fully qualified path to the driver binary file. Corresponds to the ImagePath value under service registry subkey.	
lpLoadOrderGroup	- Points to a null-terminated string that names the load ordering group of which this service is a member. Our driver does not belong to any group, so we simply pass NULL.	

lpdwTagId	- Points to a 32-bit variable that receives a unique tag value for this service in the group specified in the lpLoadOrderGroup parameter. No tag is required for us and this parameter will be NULL.
lpDependencies	- This parameter has no meaning for the driver services. It will be always NULL.
lpServiceStartName	- Pointer to a null-terminated string with account name the service should run under. If the service type is SERVICE_KERNEL_DRIVER the name is the driver object name that the system uses to load the device driver. We specify NULL as our driver is to use a default object name created by the I/O subsystem.
lpPassword	- Passwords are ignored for driver services. Should always be NULL.

Let me draw a bottom line here. In the last five parameters we always specify NULL, and you can completely forget about it. The first parameter is the handle to the SCM database. What is dwDesiredAccess for, I hope is clear too. And I think you already have guessed what are the other parameters are for. Well, they correspond to the registry keys we have analyzed above. The table below is the visual aid for you.

CreateService	Registry
lpServiceName	registry subkey name
lpDisplayName	DisplayName
dwServiceType	Type
dwStartType	Start
dwErrorControl	ErrorControl
lpBinaryPathName	ImagePath

Table 2-1. Correspondence between some parameters passing to the CreateService and the registry keys.

As you can see, not all is so black as it's painted. Let's get back to the source code.

```
        push eax
        invoke GetFullPathName, $CTA0("beeper.sys"), sizeof
acDriverPath, addr acDriverPath, esp
        pop eax

        invoke CreateService, hSCManager, $CTA0("beeper"),
$CTA0("Nice Melody Beeper"), \
                SERVICE_START + DELETE, SERVICE_KERNEL_DRIVER,
SERVICE_DEMAND_START, \
                SERVICE_ERROR_IGNORE, addr acDriverPath, NULL,
NULL, NULL, NULL, NULL
        .if eax != NULL
            mov hService, eax
```

Calling GetFullPathName function we form the complete path to the device driver file and pass it to the CreateService.

CreateService adds our driver to the SCM database, and creates an appropriate registry subkey. Look at Figure 2-1 once again. All this info was added into the registry by CreateService. If you comment the call to DeleteService out, recompile csp.asm and run it you can see exactly the same on your computer.

Don't think that using general RegXxx functions to manipulate with the registry it is possible to achieve the same result. You can add the data into the registry, but it will not appear in the SCM database.

If the specified device driver already exists in the SCM database the call to CreateService will fail. Calling GetLastError returns ERROR_SERVICE_EXISTS. If CreateService is able to successfully add the driver to the SCM database, the handle to driver is returned. This handle is required by other functions in order to manipulate the driver.

2.3.3 Starting the driver

The next function we have to call is StartService. And here is its prototype:

```
StartService proto hService:HANDLE, dwNumServiceArgs:DWORD,
lpServiceArgVectors:LPSTR
```

Parameter	Description
hService	- Identifies the opened service.
dwNumServiceArgs	- This parameter is always zero for device drivers.
lpServiceArgVectors	- Driver services do not receive any arguments. So, it should be NULL.

Now we start the driver like this:

```
            invoke StartService, hService, 0, NULL
```

The StartService function forces the system to make some actions that reminds loading common user-mode DLL. An image of the driver's file is mapped into the system address space. The driver is always mapped at arbitrary address. Then the system performs relocations within the driver image using reloc section of PE file. All references to imported symbols are fixed up. When the driver's image is prepared, the system calls an entry point of the driver, which resides in the DriverEntry routine. The main difference here is that the code of the DriverEntry routine always runs in context of the system process.

The call to StartService function is synchronous. It means it will not return until the driver's DriverEntry routine finished. If the driver initialization succeeds, DriverEntry should return STATUS_SUCCESS, and the StartService will return nonzero value. And we are back in the context of thread called StartService again, i.e. the context of our SCP.

We don't care about the value, returned by the StartService, since beeper driver has already played its nice melody and returned an error code. So, we know beforehand that the StartService will return an error.

2.3.4 Uninstalling the driver

```
            invoke DeleteService, hService
            invoke CloseServiceHandle, hService
        .else
            invoke MessageBox, NULL, $CTA0("Can't register
driver."), NULL, MB_ICONSTOP
        .endif
        invoke CloseServiceHandle, hSCManager
```

Now all we have to do is bring the system to initial state. Calling DeleteService we remove the driver from the SCM database. Strange, but it is not necessary to pass the handle of the SCM database to DeleteService. The DeleteService prototype is simple:

```
DeleteService proto hService:HANDLE
```

Parameter	Description
hService	- Identifies the service to be removed. It is necessary to have appropriate access right. We have it.

This function does not actually delete the service right away; it simply marks the service for deletion. The SCM will delete the service only when the service stops running and after all handles to the service have been closed. As we still hold the handle to the driver it's not removed from the SCM database. If you try to call

DeleteService again, it will fail. Calling GetLastError returns ERROR_SERVICE_MARKED_FOR_DELETE.

Since we don't need to communicate with the driver anymore, we must close the handle to it by calling CloseServiceHandle:

```
CloseServiceHandle proto hSCObject:HANDLE
```

Parameter	Description
hSCObject	- Handle to the driver or SCM database to be closed.

As there are no open handles to the driver now, its entry is removed from the SCM database. The second call to CloseServiceHandle closes the handle to the SCM itself.

2.4 String macros

Finally you should know what $CTA0 is. It's a macro function. It let you define ASCII string terminating with zero in read-only data section. You can use it right in the invoke macro. This macro is not sole. The file \Macros\Strings.mac contains many other useful macros to define strings with detailed explanation how to use it. Since it has nothing related with the kernel-mode driver programming I will not pay your attention to this subject anymore, but I will use such macros everywhere.