整理自: BLOG 和 MSDN

1. 术语

域

- 域的基本形式是 “[]” 加上域名，形如：[名称]。域名是不区分大小写的，所以[version]和[VERSION]被等而视之；
- 域可以处于安装文件的任何一个部位，域甚至可以被分成多分存放在安装文件中，在这种情况下，系统会自动将同名域合并成一个域；
- 子域：现有域[A],那么[A.B]称为[A]的子域；
- 指令域（指令子域）：指为完成某个指令而新增的域。例如， AddReg = 指令域1，指令域2...

Entry/入口/Item/条目/指令

- 域包含了若干条条目，这些条目后面称为指令；
- 一般来说，域中的一个有效行就是一条指令（可以用反斜杠"\"使一条指令写成多行）；
- 格式1：name=value(参数1，参数2，...)；
- 格式2：value(参数1，参数2，...)，如文件和注册表指令；
- 约定：凡是指令外面加"[]"的表示可选指令；

2. 多系统/平台支持

- 根据不同的后缀名，安装时系统会选择对应类型的指令/域；
- 多系统指令，例签名指令：

```
CatalogFile = a.cat
CatalogFile.nt = b.bat
CatalogFile.ntx86 = c.cat
CatalogFile.ntia64 = d.cat
CatalogFile.ntamd64 = e.cat
```

- 多系统域，例安装域：

```
[DDInstall]
[DDInstall.nt]
[DDInstall.ntx86]
[DDInstall.ntia64]
[DDInstall.ntamd64]
```

- 一共有5种系统后缀；
    1. 无后缀：在**Windows 2003 SP1以前**的系统上，它是**所有平台**下， **Windows 2000 及以后**系统的默认模块；在 **Windows 2003 SP1 及以后**的系统上，它是 **X86** 平台下，**Windows 2000 及以后**系统的默认模块；
    2. .nt 后缀：在 **Windows 20003 SP1 以前**的系统上，它是**所有平台**下， **Windows XP 及以后**系统的默认模块；在 **Windows 2003 SP1 及以后**的系统上，它是 **x86**平台下，**Windows XP 及以后**系统默认模块。如果第一种、第二种后缀在一个安装文件中同时满足

要求，则选择第二种后缀；

3. .ntX86：X86平台，在 Windows Xp 及以后的系统上有效，当他们和第一种或者第二种情况在同一个文件中同时满足要求时，则他们被选择，第一、二两种判断被忽略。；
4. .ntia64：安腾64平台，其他同上；
5. .ntamd64：AMD X64 平台，其他同上；

3. 指令

INF基本指令包括：**注册表、文件操作、服务指令。**

3.1 注册表指令

- 注册表指令格式：“AddReg/DelReg：子域1名[,子域2名]”
- 注册表子域指令格式：“根键，[子键]，[键值名]，[Flags]，[键值]”
  - [根键/子键]用来定位到键;
  - [键值名]用来定位到值。如果忽略键值名，则这条指令表示一个键(Key)；否则表示一个值(Value),此时[键值]代表了它的具体内容;
  - [Flags]是标志位，标识 [键值]的类型或操作的属性；
  - 例子：HKLM,Software\CY001,Parameters,0,"NULL"
    这条指令的意思是：把根键 HKLM 下的 Software\CY001 子键中包含的名称为 Parameters 的字符值的内容设置为 "NULL" 。

3.1.1 子域:根键

根键缩写含义：

- HKCR: HKEY_CLASSES_ROOT
- HKCU: HKEY_CURRENT_USER
- HKLM: HKEY_LOCAL_MACHINE
- HKU: HKEY_USERS
- HKR: 相对根键，根据当前的指令域决定注册表路径：
  - [Classinstall32]： HKR=类键=HKLM\SYSTEM\CurrentControlSet\Control\Class
  - [InterfaceInstall32]： HKR=接口键 =HKLM\SYSTEM\CurrentControlSet\Control\DeviceClasses(Class)\GUID
  - [DDInstall]： HKR=软件键 =HKLM\SYSTEM\CurrentControlSet\Control\DeviceClasses(Class)\GUID\实例ID
  - [DDInstall.HW]： HKR=硬件键=HKLM\SYSTEM\CurrentControlSet\Enum\总线\设备ID\实例ID
  - [DDInstall.CoInstallers]：HKR=协装键 =HKLM\SYSTEM\CurrentControlSet\Control\DeviceClasses(Class)\GUID
  - [服务子域]： HKR=服务键=HKLM\SYSTEM\CurrentControlSet\Services

3.1.2 子域:[软件]键值名&值

几个重要的软件键值名：

- EumPropages32：[HKR,,EumPropages32,, "prop-provider.dll,provider-entry-point"]，含有用户自定义的属性界面的 DLL 文件和入口函数；

- LocationInformationOverride：[HKR,,LocationInformationOverride,, "text-string"]，(在Windows xp 及以后系统中)一段描绘硬件设备位置的字符串，它将覆盖设备总线驱动在响应设备的IRP_MN_QUERY_DEVICE_TEXT 请求时返回的 LocationInformation 内容；
- ResourcePickerTags：[HKR,,ResourcePickerTags,, "text-string"]，设置设备的资源获取标签(Resource Picker Tags)；
- ResourcePickerExceptions：[HKR,,ResourcePickerExceptions,, "text-string"]，设置设备允许的“资源冲突”种类；

3.1.3 子域:[硬件]键值名&值

几个重要的硬件键值名：

- **DeviceCharacteristics**：[HKR,,DeviceCharacteristics,0x10001, 设备特性]：
  - [设备特性]值在 WDM.h 中定义，多值可以进行组合，然后直接在安装文件中设置。设备驱动程序调用 IoCreateDevice 函数时， PNP 管理器将把这个值和 IoCreateDevice 参数 DeviceCharacteristics 进行 OR 操作，然后再创建设备对象；
  - 常用值：

```
//分别表示可移动存储设备、软盘、只能写一次的设备(如刻录光盘)
#define FILE_REMOVABLE_MEDIA 0x00000001
#define FILE_FLOPPY_DISKETTE 0x00000004
#define FILE_WRITE_ONCE_MEDIA 0x00000008
//只读设备，如CD-ROM
#define FILE_READ_ONLY_DEVICE 0x00000002
//用户将以安全形式打开设备句柄
#define FILE_DEVICE_SECURE_OPEN 0x00000100
```

- **DeviceType**：[HKR,,DeviceType, 0x10001, 设备类型]：
  - [设备类型]在 WDM.h 中被定义成 FILE_DEVICE_XXX 形式。在总线驱动为设备创建物理设备对象 (PDO)时，从注册表中读取这个值并在调用 IoCreateDevice 时将它作为 DeviceType 参数使用。
  - 常用值：

```
#define FILE_DEVICE_CD_ROM 0x00000002 //CD-ROM
#define FILE_DEVICE_DISK 0x00000007   //磁盘
#define FILE_DEVICE_DVD 0x00000033    //DVD
#define FILE_DEVICE_MASS_STORAGE 0x0000002d //存储设备
```

- **Security**：[HKR,,Security, 0x00000, 安全描述符]：[安全描述符]为字符串类型。安全描述符的格式较繁琐。安全描述符将由 PNP 管理器负责解析并传递给设备驱动创建的所有的设备对象，包括：功能设备对象、物理设备对象，甚至过滤设备对象。
- **UpperFilters**：[HKR,,UpperFilters, 0x10000, 服务名]：标记[服务]为上层过滤驱动。PNP 管理器将在加载设备驱动时，自动检索过滤驱动，并按照顺序加载它们，以完成驱动栈和设备栈的建设。
- **LowerFilters**：[HKR,,LowerFilters, 0x10000, 服务名]：同上。
- **Exclusive**：[HKR,,Exclusive, 0x10001, 排他性]：[排他性]如果值为1，说明设备是一个排他性的设备；否则不是。用户程序通过调用 CreateFile 来获取一个内核设备对象句柄，从而使用设备提供的各种功能。以排
  他性方式创建的内核设备对象，在同一个时刻只允许用户层拥有一个打开的设备句柄(第二次调用 CreateFile 时将失败)。

3.1.4 子域:Flags

[键值]类型标识	Value	描述
FLG_ADDREG_TYPE_SZ	0x00000000	REG_SZ（字符串型）；默认类型；
FLG_ADDREG_TYPE_BINARY	0x00000001	REG_BINARY(二进制)
FLG_ADDREG_TYPE_MULTI_SZ	0x00010000	REG_MULTI_SZ（多字符串）；用逗号隔开的字符串数组，如：String1,String2
FLG_ADDREG_EXPAND_SZ	0x00020000	REG_EXPAND_SZ（扩展性字符串类型）；可以使用 %systemroot% 这样的扩展字符串；
FLG_ADDREG_TYPE_DWORD	0x00010001	REG_DWORD（双字节类型）
FLG_ADDREG_TYPE_NONE	0x00020001	REG_NONE（空）
AddReg指令操作标识	**Value**	**描述**
FLG_ADDREG_NOCLOBBER	0x00000002	只能新建：如果键值已经存在，则不做任何操作
FLG_ADDREG_DELVAL	0x00000004	删除指定的键或者键值
FLG_ADDREG_APPEND	0x00000008	如果键值不存在，则新建：如果存在，则判断新添加的内容是否已经存在于原内容中，如果存在，则不操作，如果不存在，则将新内容添加到原内容后面； 注意：此值必须与 FLG_ADDREG_TYPE_MULTI_SZ 一起使用，否则将被忽略
FLG_ADDREG_KEYONLY	0x00000010	只创建/删除子键：忽略参数“键值名”与“键值内容”
FLG_ADDREG_OVERWRITEONLY	0x00000020	只更新，不新建。如果键值不存在，则不做任何操作：如果键值存在，则更新原内容
FLG_ADDREG_64BITKEY	0x00001000	(在 windows XP 及以后系统中)一般将此 Flag 与其他 Flag 值合用。当设置此值时，表明所有操作针对64位注册表进行；否则，所有操作针对本地注册表进行。
FLG_ADDREG_KEYONLY_COMMON	0x00002000	(在 windows XP 及以后系统中)意义与 FLG_ADDREG_KEYONLY 相同，不同之处在于这个值也能用于 DelReg 指令中
FLG_ADDREG_32BITKEY	0x00004000	(在 windows XP 及以后系统中)一般将此 Flag 与其他 Flag 值合用。当设置此值时，表明所有操作针对32位注册表进行；

		否则，所有操作针对本地注册表进行。
DelReg指令操作标识	**Value**	**描述**
FLG_DELREG_KEYONLY_COMMON	0x00002000	删除整个子键
FLG_DELREG_32BITKEY	0x00004000	所有操作针对 32 位注册表：如果不设置，则针对本地注册表(64位系统上本地注册表是64位的)
FLG_DELREG_MULTI_SZ_DELSTRING	0x00018002	在一个类型为 REG_MULTI_SZ 的键值中，删除字符串列表中的某些字符串。比如：某字符串列表包含 String1,String2 ，现指定删除 String1 ，则键值不会被删除，但其内存仅存 String2

3.1.5 BitReg指令

- 这个指令很少被用到。它能够改变类型为 REG_BINARY 的键值的指定位的值。
- 指令格式：BitReg = 子域[,子域2]...
- 子域指令格式：根键，[子键]，键值，[Flags]，位掩码，字节位置
  - 其中参数 Flags 是一个可选参数，在默认情况下值为0。它有3种可选值：0表示清除，将指定位设置为0；1表示设置，将指定位设置为1；
  - 例：HKLM,Software\Sample,Attributes,0x01,0xFF,0。把类型为 REG_BINARY 的键值 Software\Sample\Attributes 的第一个字节(最后的参数0)的所有位(0xFF)都设置为1

3.2 文件操作指令

3.2.1 CopyFiles

- CopyFiles 指令可以将一个指定的源文件拷贝到目标地址；也可以通过子域将一组文件拷贝到目标地址。
- 格式如下：CopyFiles = @文件名,文件子域1[,文件子域2]...；以@开头的参数，指定一个文件；否则指定一个文件子域，文件子域中可以包含一个或多个文件。
- 文件子域指令格式：

```
[文件子域]
目标文件名1[,[源文件名][,[废除][,flags]]]
目标文件名2....
```

3.2.2 DelFiles

- DelFiles 执行 CopyFiles 的反操作，指令格式基本相同。
- 文件子域指令格式:

```
[文件子域]
目标文件名1[,[废除],[废除],Flags]
目标文件名2....
```

- Flags:

- DELFLG_IN_USE(0x00000001): 如果删除过程中文件正被使用，则将等到使用的文件内关闭后在删除
- DELFLG_IN_USE1(0x00010000): 与 DELFLG_IN_USE 相同，只能用于 Windows 2000 以后的系统中

3.2.3 RenFiles

- RenFiles 可以对（多个）目标文件进行重命名。
- 指令格式：RenFiles = 文件子域1[,文件子域2]...
- 子域指令格式：

```
[子域名]
新文件名1，源文件名1
新文件名2...
```

3.3 服务指令

- 服务包括：系统服务和功能驱动；
- 服务的信息被保存在注册表的服务键中(即 HKLM\System\CurrentControlSet\Service\服务名)
- 服务指令就是对服务的操作，包括：安装和卸载；

3.3.1 AddService

- AddService 指令向系统中添加一个服务；
- 指令格式：AddService = 服务名，[Flags],服务子域名[,事件 Log 子域名 [,[事件 Log 类型][,事件名]]]；
- [Flags]定义在Setupapi.h中，查看附录X.I,也可参见MSDN；
- [服务子域名]：指向服务子域，子域中对新建的服务进行值配置；
- 最后三个可选参数，为设备驱动向系统注册事件Log 入口，通过系统EventViewer查看；
- 服务子域指令格式：

```
DisplayName = 服务显示名称          ;友善的名称,方便定位
[Description = 描述字符串]
ServiceType = 服务类型              ;对于内核驱动,必须为1;其他值定义在Wdm.h/Ntddk.h中
StartType = 启动类型                ;控制启动的时机,对于设备驱动一般为SERVICE_DEMAND_STA
TR(0x3);其他值参见MSDN;
ErrorControl = 错误控制级别         ;错误控制级别，当服务运行发生错误时，将根据这个值采
取一定的错误控制;
ServiceBinary = .sys 文件路径       ;镜像文件路径，此处当然是 .sys 文件
[StartName = 驱动对象名]            ;将创建的驱动对象(即 DRIVER_OBJECT 对象)名称，如果不
设置这个参数，则驱动对象以服务为名
[Security = 安全描述符]             ;指定一个安全描述符
[AddReg = ...]
```

3.3.2 DelService

- DelService 指令把一个服务从系统中删除；
- 指令格式：DelService = 服务名[,[Flags][,[EventLogType][,EventName]]
- [Flags] 只有一个可用值：0x00000200(SPSVCINST_STOPSERVICE)，表示在删除服务前，先要停止服务。

- 最后两个可选参数用来删除事件 Log 入口。

4. 域

- 域是很松散的，任一个域都可以任意散布在安装文件的任意位置——也就是说，只要存在就可以了；
- 将域分成如下几组进行讲解：
  - 版本域：版本信息；
  - 文件域：文件操作；
  - 控制域：安装控制；
  - 字符串域：字符串操作；
  - 厂商域：厂商/产品信息；
  - 类安装域：设备类、设备接口类；
  - 设备安装域：设备信息；安装设备驱动（这是最复杂的部分）；
- 厂商/产品/设备的关系：一个厂商发布多个系列产品，而在每个系列产品中包含了多个不同型号设备。

4.1 版本域

- 唯一必选的域
- 版本域提供了驱动包的版本信息
- 解释：

```
[Version]
Signature = "签名"                          ;记住只用"$WindowsNT$"或"$Chicago$",含义相同:"本安装文件支持所有的操作系统"
DriverVer = mm/dd/yyyy[,w.x.y.z]            ;驱动版本，格式是：日期(月/日/年)+版本号（数字的范围是[0,65534]）
[Class = 设备类名]                          ;设备（安装）类名称；尽量使用系统预定义；所有的安装类都放于HKLM\SYSTEM\CurrentControlSet\Control\Class中
[ClassGuid = { GUID}]                       ;设备类 GUID。GUID，16字节长，或32个十六进制数
;DDK头文件DEVGUID.h定义了标准设备类的GUID
[Provider = 作者]                           ;INF 文件作者，可能是设备厂商或者第三方公司
;下面几个指令是签名指令
[LayoutFile = xxx.inf [,xxx.inf]... ]       ;(Win 2K and later)
[CatalogFile = xxx.cat]                     ;catalog 文件，数字签名文件
[DontReflectOffline = 1]                    ;(Vista and later)
[PnpLockDown = 0|1]                         ;(Vista and later)
[DriverPackageDisplayName = %驱动包描述符%]
[DriverPackageType = 类型]
```

4.2 文件域

- 与文件操作相关的域有四个：
  1. 源磁盘域([SourceDiskNames])
  2. 源文件域([SourceDiskFiles])
  3. 目的域([DestinationDirs])
  4. 文件列表子域(是目的域的子域)
- 前两个域总是和文件拷贝联系在一起，不存在文件拷贝指令(CopyFiles 指令)，这两个域就不必存在；

- 后两个域则当存在文件拷贝指令、文件删除指令、文件重命名指令时，都必须存在。

4.2.1 源磁盘域

- 源磁盘可以是:光盘/软盘/本地文件夹
- 格式如下：

```
[SourceDiskNames]
磁盘ID1 = 磁盘描述[,[tag-or-cab 文件],[unused],[路径][,Flags][,tag 文件]]
磁盘ID2...
```

  - [tag 文件]只有在 Windows XP 以后的系统中才能用，否则将被忽略;
  - [tag-or-cab]：**tag** 是标签的意思， tag 文件相当于在光盘上贴一个记事贴。对于光盘等移动存储介质，可以指定一个 .tag 文件，当移动存储介质插入系统后，系统在其目录下检索是否存在此 .tag 文件，如果没有，则认为此介质不是系统想要的，将提醒用户找不到驱动文件。.tag 文件没有复杂的格式，仅仅依靠文件来判断。cab 是 cabinet（储存柜） 的缩写，里面能储存很多东西，这里储存的是驱动包文件(.sys 、 .dll 、.exe 等)。如果设置了 cab 文件，则系统找不到指定的驱动文件时，会再去 cab 文件中寻找。其实操作系统用到了很多 cab 文件，这应该是 zip 或 rar 类型的压缩格式。历史上 tag-or-cab 文件是混用的，既以其文件名充当 tag 文件，又以其内容做 cab 文件。在 Windows XP 以后为了解决这个问题，就多了最后一个"tag 文件"参数，而把 tag-or-cab 文件专门用作 cab 文件。举一个例子：
- 例：

```
[SourceDiskNames]
1 = "Disk1","MyCab.cab",,"Driver\MyDriver",0x10,"MyTag.tag"
```

含义：在名为 Disk1 的驱动安装盘中，根目录或 Driver\MyDriver 子目录下有一个名为 MyTag.tag 的标签文件，若此文件不存在，则可判定为非法磁盘。此外，如果有文件在驱动子目录中找不到，则应在档案文件MyCab.cab 中继续搜索，档案文件可能位于驱动子目录下，也可能位于磁盘根目录下

4.2.2 源文件域

- 指示安装文件位置
- 指令格式如下：

```
[SourceDiskFiles]
文件名1 = 磁盘ID[,[路径][,大小]]
文件名2...
```

  - 参数"路径"相对于磁盘指令中的"路径"参数，就是上例中出现的 Driver\MyDriver ；
  - 参数“大小”用来表示此文件大小(未压缩状态)，单位为字节，用来对指定文件做有效性判断。
  - 例:"file1=1,sys," 的意思是：在磁盘1驱动子目录下的 sys 子目录中，有一个名为 file1 的文件。

4.2.3 目的域

- 目的域用来定义所有文件操作的目标目录。

- 只要安装文件中含有 CopyFiles 、 DelFiles 或 RenFiles 指令，就一定要有目的域。
- 格式如下：

```
[DestinationDirs]
[DefaultDesDir = 目录 ID[,子目录]]      ;默认的目标
[文件列表子域名1 = 目录 ID[,子目录]]
[文件列表子域名2 = 目录 ID[,子目录]]
```

- 目录 ID 用来将系统常用的一些目录用数字来代替；从 16384(0x4000)开始到 32767(0x7FFF)之间的值，是Shell保留目录；常用的目录ID如下,更多查看附录X.II；
  - 01 SourceDrive:\pathname(当前的安装文件所在目录)
  - 10 Windows 文件夹，等同于： %SystemRoot%
  - 11 系统文件夹，等同于： %SystemRoot%\System32
  - 12 Drivers 文件夹，等同于： %SystemRoot%\System32\drivers
  - 17 INF 文件夹，等同于： %SystemRoot%\Inf
  - 24 系统盘根目录。操作系统安装在哪里目录 ID 即代表哪个盘，如 C:\
  - 30 启动盘根目录。也就是引导分区所在的那个磁盘，或者说是物理上的第一个分区盘。若不出意外，总是等于 C:\。
  - 他不一定等于目录 ID24 所代表的目录，另外，系统可以安装在非启动分区上
  - 50 系统文件夹(!=11)，等同于：%SystemRoot%\system
  - 53 用户目录
  - 54 系统引导文件 Ntldr.exe 和 Osloader.exe 所在的目录。一般情况下，等同于 30 所表示的目录
- 示例

```
[Files1]
123.sys
[Files2]
123.inf
[Files3]
123.exe
[DestinationDirs]
DefaultDesDir = 10,system32    ;它等价于：  DefaultDesDir = 11
Files1 = 11,Drivers
Files2 = 12
```

上面共有3个文件列表子域，最后的运行结果是：文件 123.inf 和 123.sys 将被拷贝到 "\system32\Drivers" 目录，而文件 123.exe 将被拷贝到默认的 system32 目录。

4.3 控制域

- [ControlFlags]域称为控制域，它指示了安装过程中必须遵循的一些规定。
- 域指令格式如下：

```
[ControlFlags]
;其作用是让此 INF 文件中的某设备不在 "添加硬件向导"中显示
;使得用户无法对这些设备进行手动预安装(非 PNP 设备的安装方式)
;如果指令值为*，则包括了此 INF 文件中的全部设备
ExcludeFromSelect = * |
ExcludeFromSelect = 设备 ID[,设备 ID] ...]

;只拷贝文件，不安装驱动
```

```
;先将驱动文件拷贝到系统中，当下次设备插入时，再安装驱动
[CopyFilesOnly = 设备 ID[,设备 ID] ...]

;迫使安装过程发生在用户交互环境下，系统将显示用户界面。
[InterfaceInstall = 设备 ID [,设备 ID] ...]

;这个指令非常新，只用于 Windows 7 及以后的操作系统中,他一般用来进行网络更新，很酷的功能
;比如有一块显卡设备，厂商为了尽快推出产品占领市场,不可能在第一个版本中实现全部功能，只能先发布一个基本版本，然后不断对此进行更新；而使用此命令后
;PNP 管理器会在安装过程中到网络上检查并下载最新的驱动，通知用户更新，这使得显卡在每次安装驱动时，都能够保持版本最新
[RequestAdditionalSoftware = *] |
[RequestAdditionalSoftware = 设备 ID [,设备 ID] ...];(Windows 7 and later)
```

- 使用最广泛的是 ExcludeFromSelect 指令，也可以称为“排除指令” -- 排除了通过“添加硬件向导”安装设备的可能性。
- "添加硬件向导"是一个很有趣的安装模块，所有通过他安装的设备驱动，其总线都是系统本身，即称为 Root 虚拟总线。这个 Root 虚拟总线是整个计算机的根总线。所有的非 PNP 设备，由于没有更合适的总线驱动，所以都是以它为总线的。

4.4 字符串域

- 在这里，字符串定义叫作字符令牌；
- 指令格式：%令牌名% = "..."
- 字符串域的另外一个特点是，能够提供多国语言的国际化支持。所谓国际化支持，就是一个令牌可以对应着多个语言版本的字符串。加入国际化支持后的字符串域，定义成格式：[String.语言ID]
  - 示例：

```
[Strings.409] ;英语 ID 为 0x409
Love = "I Love You" ;英文爱情
[Strings.804] ;中文 ID 为 0x804
Love = "我爱你" ;中文爱情
```

  - 安装程序在执行此安装文件时，将根据系统语言寻找最佳匹配，在中文系统上安装时会使用“我爱你”，在英文系统上安装时则使用 “I Love you”。这种语言匹配是自动进行的。
- 字符串令牌拼接, 示例:

```
;下面是4个令牌连用的例子
;域名也是字符串形式的
;字符间是有空格的
AddReg = %A% %B% %C% %D% ;等价于:  AddReg = "我是谁？"

[我是谁？] ;AddReg 的子域，即 "%A% %B% %C% %D%"
HKLM,,,,
;......
[Strings]
A="我"
B="是"
C="谁"
D="?"
```

- 在使用令牌的情况下，安装文件中字符串长度最长为 512 个字符(Windows XP及以前系统)或4096个字符(Windows Vista 及以后系统)。将令牌置换成实际内容后(可能包括多个字符串拼接在一起)，实际内容的最大长度为4096 个字符(所有系统)。以上讲的长度都包含了结尾的 NULL

字符，也就是说，实际允许的最大长度都要减一。

4.5 厂商/产品域

示例：著名的电脑厂商有联想、Dell、苹果等。联想的牌子有 ThinkPad 、 IdeaPad 等； Dell 有 OptiPlex 、 Inspiron 等；苹果有 iMac 、 MacBook 等。
现在厂商为：联想、 Dell 和苹果。每个厂商各有一个与“电脑”相关的产品域。而以联想为例，它的产品型号包括： IdeaPad 、 ThinkPad 等。
安装模块在安装文件中依次搜索匹配的设备，寻找过程类似于下面所示：找厂商 找产品 找设备
生产厂商1---|
. |------>产品A---|
. . |------>设备甲---->安装...
. . .

4.5.1 厂商域

- 厂商域[Manufacturer]定义了一个或多个厂商信息，这些厂商的名下又将有一类或多类产品信息。
- 示例：

```
[Manufacturer]
;指令名(左)是厂商名;
;指令值(右)是产品域名，通过它来安装厂商名下的多个设备(产品)
"Dell" = Dell

;指令值中可以包括多个 OS 版本信息，比如 NTx86 ，那么对应的就会多一个安装域
"Lenovo" = Lenovo,NTx86
"SONY"= SONY

;下面是产品域，联想有两个产品域；戴尔、苹果只有一个
[Lenovo]
"ThinkPad" =
"IdeaPad" =
[Dell]
;...
[Apple]
;...
```

4.5.2 产品域

- 用来标识一系列被安装的设备，可以把产品域当成一条产品线，产品线上有多个不同型号的产品(设备)。
- 指令格式：

```
[产品域名称.OS版本]         ;只有 Windows XP 及以后版本才能识别 OS 版本信息
;指令名(左)表示设备名称     ;指令值(右)表示设备安装域，非常重要
设备名1 = 设备安装域[,硬件 ID][,兼容 ID ...]    ;硬件、兼容 ID 标识硬件本身，只会符合设备 ID 或 兼容 ID 的才能够进入“设备安装域”进行设备安装
设备名2...
```

- 示例，以 CY001 开发板为例，厂商 ChinaHearing 旗下有一款 CY001 产品， CY001 则有两个型号(Original 和 New)：

```
[Manufacturer]
%MfgName%=CY001,NTx86    ;名为 CY001 的产品

;CY001 的产品域旗下包含多个设备，两个设备共用同一个设备安装域
[CY001.NTx86]
"Original"=CY001.Dev,USB\VID_04B4&PID_8613
"New"=CY001.Dev,USB\VID_04B4&PID_1109
```

后面会讲解设备域(或设备安装域)，也被称做 DDInstall 域。

4.6 设备类安装域

- 系统将所有注册的设备类保存在下面的注册表路径
  中：HKLM\SYSTEM\CurrentControlSet\Control\Class
- 通过设备类安装域把一个新的设备类及其相关内容安装(或注册)到系统中。一般仅在初次安装的时候起作用：安装组件会检查 Version 域中的Class与ClassGUID 值，如果他们已经在系统中注册过，则继续安装设备驱动；否则，需要首先检查安装文件中是否有设备类安装域，如果存在，则先安装设备类，然后在安装设备驱动；如果找不到设备类安装域，则设备驱动安装无法进行。
- 系统不对用户自定义设备类别的操作做任何限制，谁都可以定义新设备类。
- 格式：
  [Classinstall32]
  …
  [ClassInstall32.Services] ;服务子域，用来为设备类安装服务。一般我们都只为某个设备安装服务，很少会针对设备类安装服务。
  …
- 示例/解释：

```
[Classinstall32]
AddReg = testAddReg, ...    ;重要的唯一的必选指令,为新创建的设备类在注册表中添加必要的信息,
[DelReg=子域,...]
[CopyFiles = @文件名 | 文件子域, ...]
[DelFiles=文件子域, ...]
[RenFiles=文件子域, ...]
[AddProperty = 子域, ...](Windows Vista 及以后系统)
[DelProperty = 子域, ...](Windows Vista 及以后系统)
[BitReg = 子域, ...]
[UpdateInis = 更新子域, ...]
[UpdateIniFields=更新子域(Field), ...]
[Ini2Reg = 子域, ...]

[testAddReg]
HKR,,,0,%ClassName%        ;设备类的名字(可读名称)
[HKR,,Installer32,,"class-installer.dll,class-entry-point"] ; 类安装器
[HKR,,EnumPropPages32,,"prop-provider.dll,provider-entry-point"];自定义的属性页
;类图标，在设备管理器中显示，必须设置 ;正值：图标从用户文件中取； 负值：图标从系统文件中取
;可使用的系统图标 ID 包括： 0、2、3、5、6、8~10、14、15~18
HKR,,Icon,,"icon-number" ; 类图标
;下面3个指令可选 ;1，代表设置属性； 其他值，代表不设置
[HKR,,SilentInstall,,1]
[HKR,,NoInstallClass,,1]
[HKR,,NoDisplayClass,,1]
```

    - Icon 作用是为这个新建类选择一个系统图标。如果 Icon 值为正值，那么系统将从安装器文件中获取；如果类安装器不存在，就从属性页文件中获取；如果二者都不存在，则由系统默认指定。如果 Icon 值为负值，则 Icon 为系统预定义的图标，图标内容由具体的值确定。
- 示例，CY001 开发板设备注册自定义的设备类：

```
[Version]
Signature = "$WINDOWS NT$"
Class = CY001 Sample ;自定义的安装类，即下面的类 GUID 值
ClassGuid = {78A1c.....}
[ClassInstall32]
AddReg = SampleClassReg
[SampleClassReg]
HKR,,,0,%ClassName%
HKR,,Icon,,-5
[Strings]
ClassName = "CY001: USB驱动开发板设备"  ;类友好名称(Friendly Name),此名称将在设备管理器中显示，可以包含中文
```

包含上面内容的 INF 文件被首次安装后，就能在设备管理器中看到 CY001 的设备类了。

4.7 接口类安装域

- 接口类安装域能够向系统中注册一个或多个新的设备接口类。
- 使用接口类安装域是静态注册设备接口类；通过在程序中调用内核函数 IoRegisterDeviceInterface 是动态注册。
- **一般接口类安装域总是和设备安装域中要讲到的“接口子域”一起使用，接口子域在接口类下创建实例，接口实例名可用作与底层设备交互的接口。(不明觉厉)**
- 格式如下：

```
[InterfaceInstall32]
{类 GUID}=安装子域[,Flags]
[安装子域]
AddReg = 注册表子域[,注册表子域] ...
[AddProperty = 属性子域[,属性子域] ...]
[CopyFiles = @文件 | 文件注册表子域[,文件列表子域] ...]
[DelReg = 注册表子域[,注册表子域] ...]
[DelProperty = 属性子域[,属性子域] ...]
[BitReg = 注册表子域[,注册表子域] ...]
[DelFiles = 文件列表子域[,文件列表子域] ...]
[RenFiles = 文件列表子域[,文件列表子域] ...]
[UpdateInis = update-ini-section[,update-ini-section]...]
[UpdateIniFields = update-inifields-section[,update-inifields-section]...]
[Ini2Reg = ini-to-registry-section[,ini-to-registry-section]...]
```

- 示例，仅把一个友好名称为 "DevInterface Sample" 的接口注册到系统中，此外并不做任何事情:

```
[InterfaceInstall32]
%Guid%=NewInterface

[NewInterface]
AddReg=InterAddReg

[InterAddReg]
```

```
HKR,,FriendlyName,,%IntfDesc%

[Strings]
Guid = "{028748C8.....}"
IntfDesc = "DevInterface Sample"
```

4.8 设备安装域

- 设备安装域用来安装设备，它没有固定的名称，其名称应由产品域中的指令指定，就像上面的 CY001.Dev 一样。我们总是用[DDInstall]来代表设备安装域。
- 一个设备安装域可以为多个设备安装驱动，如上例中的 Original 和 New 两个设备就共用了 CY001.Dev 这个安装域。
- 设备安装域的注册表当前路径(即 HKR)为当前设备的软件键: HKLM\SYSTEM\CurrentControlSet\Control\DeviceClasses\GUID\实例 ID ，其中实例 ID 是从0开始的4位整数，如"0001"。

4.8.1 安装域指令格式

```
[DDInstall]
;文件拷贝
[CopyFiles = @文件名 | 文件列表域[,文件列表域] ...]

;把 INF 文件拷贝到 Windows\Inf 目录下，并以 OEMxx.inf 的形式重命名
[CopyINF = filename1.inf[,filename2.inf]...]

;在注册表中添加内容
[AddReg=注册表域[,注册表域]...]

;Include 和 Needs 总是合用
;把其他的 INF 文件包含到安装过程中(类似#include)
;其语义是：把被 Include 文件中的 Needs 域包含进当前 INF 文件中
[Include=filename1.inf[,filename2.inf]...]
[Needs=inf-section-name[,inf-section-name]...]

;删除文件;重命名文件;删除注册表中信息;修改注册表(REG_BINARY)
[DelFiles=文件列表域[,文件列表域]...]
[RenFiles = 文件列表域[,文件列表域] ...]
[DelReg = 注册表域[,注册表域] ...]
[BitReg = 注册表域[,注册表域] ...]

;添加/删除/修改“开始”菜单内容。
[ProfileItems=属性域[,属性域]...]

;UpdateInis 系列指令很少被使用;它能够指定一个 INI 文件名称，并通过指令修改这个 INI 文件中的内容
;UpdateIniFields 则指定了这个 INI 文件中将被修改的内容(域名)
;实际上，这令 INF 文件达到了动态设置(Set)信息的目的;(可惜的是， INF 文件无法动态获取(Get)信息)
;Ini2Reg 则是将 INI 文件中的指定内容拷贝到注册表中(相对于 HKR 路径)
[UpdateInis = update-ini-section[,update-ini-section]...]
[UpdateIniFields = update-inifields-section[,update-inifields-section]...]
[Ini2Reg = ini-to-registry-section[,ini-to-registry-section]...]

;注册一些 DLL
[RegisterDlls=register-dll-section[,register-dll-section]...]
;注销一些 DLL
```

```
37. [UnregisterDlls=unregister-dll-section[,unregister-dll-section]...]...
```

- RegisterDlls/UnregisterDlls指令将调用Windows\System32\avtapi.dll 的 DllRegisterServer/DllUnregisterServer接口函数来注册/注销 DLL 形式的 OLE 控件组件，示例：

```
1. [DDInstall]
2. RegisterDlls = DialerRegSvr
3. UnregisterDlls = DialerRegSvr
4. [DialerRegSvr]
5. 11,avtapi.dll,1
```

4.8.2 服务子域

- 其名如： [DDInstall.Services]，指令格式就是普通的服务子域格式，参见 3.3服务指令。

4.8.3 硬件子域

- 其名如： [DDInstall.HW]，硬件子域是可选的；
- 硬件子域的主要工作是注册表操作。当执行硬件子域时，注册表当前路径(也就是 HKR)为当前设备的硬件键，具体键名称为：HKLM\SYSTEM\CurrentControlSet\Enum\总线\设备ID\实例ID
- 最典型的用途是用来为设备添加上、下层过滤驱动；
- 设备过滤驱动的注册，是通过在注册表中插入 UpperFilter 和 LowerFilter 键值来完成的。
- 子域的指令格式如下：

```
1. [DDInstall.HW]
2. ;注册表操作;添加;删除;修改(REG_BINARY)
3. [AddReg = 注册表子域[,注册表子域]...] ...
4. [DelReg = 注册表子域[,注册表子域] ...] ...
5. [BitReg = 注册表子域[,注册表子域] ...] ...
6.
7. ;Include 和 Needs 联合使用
8. [Include=filename1.inf[,filename2.inf]...]
9. [Needs=inf-section-name[,inf-section-name]...]
```

- 示例，为一个光驱设备添加上层和下层过滤驱动：

```
1. [MyDDInstall.HW]
2. AddReg=MyAddReg
3. ;为光驱设备添加一个上层和下层过滤驱动
4. [MyAddReg]
5. HKR,,"UpperFilters",0x00010000,"redbook"
6. HKR,,"LowerFilters",0x00010000,"imapi"
```

4.8.4 协安装器子域

- 其名如：[DDInstall.CoInstallers]。
- 协安装器模块都是以动态链接库(如WdfCoInstaller01011.dll)的形式存在的，所以协安装器子域的作用就是把一个或多个 DLL 文件记录到注册表中，并把他们拷贝到相关目录下。
- 当执行协安装器子域时，注册表当前路径(即 HKR)为软件键。**（需要确认）**
- 子域的指令格式如下：

```
1. [DDInstall.CoInstallers]
```

```
;AddReg 和 CopyFiles 指令为必选指令
AddReg=注册表子域[,注册表子域]...
CopyFiles=@文件名 | 文件列表子域[,文件列表子域]...

;Include 和 Needs 联合使用
[Include=filename1.inf[,filename2.inf]...]
[Needs=inf-section-name[,inf-section-name]...]

;文件、注册表操作指令
[DelFiles = file-list-section[,file-list-section]...]
[RenFiles = file-list-section[,file-list-section]...]
[DelReg = del-registry-section[,del-registry-section] ...]
[BitReg = bit-registry-section[,bit-registry-section] ...]

;INI 操作指令
[UpdateInis = update-ini-section[,update-ini-section]...]
[UpdateIniFields = update-inifields-section[,update-inifields-section]...]
[Ini2Reg = ini-to-registry-section[,ini-to-registry-section]...]
```

- 示例，仍以 CY001 开发板为例，所有的 WDF 驱动器程序都必须注册系统提供的 WDF 协安装器 ：

```
[CY001.Dev.NT.CoInstallers]
AddReg = CoInstaller_AddReg
CopyFiles = CoInstaller_CopyFiles

;把协安装器文件(WdfCoInstaller01007.dll)拷贝到系统中
[CoInstaller_CopyFiles]
WdfCoInstaller01007.dll

;在注册表中添加协安装器信息
[CoInstaller_AddReg]
HKR,,CoInstallers32,0x00010000,"WdfCoInstaller01007.dll,WdfCoInstaller"
```

4.8.5 接口子域

- 其名如：[DDInstall.Interfaces]。接口子域用来定义设备接口。
- 系统的所有设备接口注册在注册表的如下路径:..\CurrentControlSet\Control\DeviceClasses\(GUID)
- 用户程序通过其名称识别底层设备，系统中的设备名称分为两类：即符号连接和设备接口.符号链接是一个可读字符串,比如"SFilter" 、"CY001"等；设备接口是由指定的接口类 GUID 字符串与设备实例 ID 以某种方式拼接成的不可读字符串，它能最大程度地保证其名称的系统唯一性。
- 格式如下：

```
[DDInstall.Interfaces]
;向系统注册一个设备接口
AddInterface = {接口类 GUID}{,[参考字符串] [,[指令子域]]} ...

;Include 和 Needs 联合使用
[Include=filename1.inf[,filename2.inf]...]
[Needs=inf-section-name[,inf-section-name]...]
```

  - AddInterface 指令首先在系统中检查接口类 GUID 所代表的接口类是否存在，如果不存在，则忽略这条指令；否则，按照指令内容在此接口类下创建一个接口实例，此接口实例名由接口类 GUID 、参考字符串、设备实例 ID 等内容共同构成。如果接口类不存在，则

AddInterface 指令将不会执行，遇到此情况时，应在安装文件中配合加入[InterfaceInstall32]接口安装域。
    - AddInterface 指令的第三个可选参数是一个指令子域，可以在这个子域中使用 AddReg 指令为新的接口设置注册表内容。
- 示例如下：

```
[Device.Interfaces]
AddInterface = %GUID%, %REFERENCE%
[Strings]
GUID = "{6994ad04-93ef-11d0-a3cc-00a0c9223196}" //接口类 GUID
REFERENCE = "NewInterface" //参考字符串
```

4.8.6 厂商默认配置子域*

- 其名如：[DDInstall.FactDef]。 FactDef 是 Factory Default(厂商默认)的缩写。
- 所谓厂商默认，就是设备在出厂时，厂商对设备需占用的硬件资源所进行的默认配置，这些资源包括硬件 IO 端口、 DMA 通道中中断请求号。硬件产品一般都有”厂商默认配置 “，典型的如 BIOS 。
- 默认配置子域只对非 PNP 设备有用，为他们静态分配硬件资源，如 PCI 设备； PNP 设备不需要这个子域。因为 PNP 设备总是能够动态的获取硬件资源。
- 指令格式如下：

```
[DDInstall.FactDef]
LogConfig=配置子域[,...]

[配置子域]
;ConfigPriority 用来描述这段厂商默认配置在性能上的“重点”
;SUBOPTIMAL 表明性能最差(可能占用资源最少)
ConfigPriority=Priority_Value

;DMA 配置
[DMAConfig = [DMAattrs:]DMANum]

;IO 端口配置，起始地址从 2F8 开始，比如
;IOConfig = 2F8-300
[IOConfig = io-range]

;内存配置，起始地址从 D0000 开始，比如
;MemConfig = D0000 - D5000
[MemConf=mem-range]

;中断配置(IRQ)，可以分配一个中断号
[IRQConfig=[IRQattrs:]IRQNum]
```

- 示例，来自 WDK 中src\audio\sb16\mssb16.inf，这是一个 PCI 声卡驱动：

```
[WDMPNPB003_Device.FactDef]
ConfigPriority = NORMAL ;性能特征：普通
;IO端口配置：此声卡占用多段 IO 端口
IOConfig = 220-22F
IOConfig = 330-331
IOConfig = 388-38B
;中断号为5
IRQConfig = 5
```

```
9.
10.  ;DMA 配置，占用了 DMA 控制器的1、5 通道
11.  DMAConfig = 1
12.  DMAConfig = 5
```

4.8.7 逻辑优先配置子域*

- 其名如：[DDInstall.LogConfigOverride]，是 Logical Configuration of Override 的缩写，“优先配置”。
- 和 FactDef 子域不同的是，这个子域为 PNP 设备服务。
- 格式如下：

```
1.  [DDInstall.LogConfigOverride]
2.  ;性能重点
3.  ConfigPriority = Priority_Value[,Config_Type]
4.  ;下面几个域与 FactDef 相同
5.  [DMAConfig = [DMAattrs:]DMANum[,DMANum]...]
6.  [IOConfig = io-range[,io-range]...]
7.  [MemConf=mem-range[,mem-range]...]
8.  [IRQConfig=[IRQattrs:]IRQNum[,IRQNum]...]
9.  ;PcCardConfig 有两个作用
10. ;1. 配置硬件的 CardBus 寄存器
11. ;2. 在系统内存空间中创建不超过两个内存窗口
12. ; 将这些内存窗口映射到设备的属性空间中，以提高设备内存的存取速度
13. [PcCardConfig = ConfigIndex[:[MemoryCardBase1][:MemoryCardBase1]][(attrs)]]
14. [MfCardConfig=ConfigRegBase:ConfigOptions[:IoResourceIndex][attrs]...]
```

如果读者有 PS/2 鼠标键盘，他们都是 PNP 支持的设备，从设备管理器中可以查看到其资源利用情况。

4.9 默认安装域(未整理)*

我们把默认安装域分成主域和子域两个部分来讲解。
1. [DefaultInstall]主域
默认安装域是很多人的最爱，正是它使得 INF 文件能够具有神奇的右键“安装”功能。右键安装的原理是执行了 Shell 调用，这个 Shell 调用记录在注册表中，如下所示：
HKCR\inffile\Shell\Install\Command
在用户对某安装文件进行“安装”操作后，系统将把此安装文件路径作为参数，并调用上述注册表路径下的 Shell 命令。 Windows Vista 以前系统默认定义的 Shell 命令的内容如下：

```
1.  %SystemRoot%\System32\rundll32.exe \
2.  setupapi,\                  //SetupAPI.dll
3.  InstallHinfSection\         //函数名
4.  DefaultInstall\             // INF 中 DefaultInstall 域将被执行
5.  132 \                       //执行模式
6.  %1                          //安装文件路径
```

读者可以将上面的代码放到控制台程序中运行，不过要将最后一个参数 "%1"替换成实际路径，以观察其效果。 Rundll32 作为动态链接库的公用宿主之一，可以直接调用链接库中符合定义的接口函数。它的好处是不用自己写宿主程序，局限性是只能调用某类型特定的接口函数。
上面脚本命令的意思是:将字符串 "DefaultInstall 132 %1"作为参数调用 SetupAPI.dll 库中的函数 InstallHinfSection，翻译成代码就是下面这样：

```
InstallHinfSection{
NULL,NULL,
TEXT{"DefaultInstall 132 Inf 路径"},//命令行
0};
```

神奇的 InstallHinfSection 函数能够以指定的模式，执行指定 INF 文件中的某个指定域中的所有指令。他也是 SetupAPI.dll 的导出函数，属于 SetupAPI 接口系列。
上面连讲三个“指定”，特别是最后一个“指定域”，为的是告诉大家一点：INF 文件中的执行域是通过参数指定的，可以不是 DefaultInstall--但系统总是把 DefaultInstall 作为它的默认安装域。
Windows Vista 以后系统的 Shell 命令变成了下面这个样子：
%SystemRoot%\System32\InfDefaultInstall.exe "%1"
这里未对 InfDefaultInstall.exe 文件做细致分析，但查看其镜像文件后发现它引入(Import)了 InstallHinfSectionW 接口，故而猜测他并没有使用更新的执行方法。但笔者个人更喜欢 InfDefaultInstall.exe 方式，因为首先它很简单，其次使用 Rundll32 多少有些冷冰冰和莫名其妙的感觉。下面看这个域的指令格式。

```
[DefaultInstall]
;拷贝文件、删除文件、重命名文件
[CopyFiles = @文件名 | 文件子域[,文件子域2] ...]
[DelFiles = 文件子域[,文件子域2] ...]
[RenFiles = 文件子域[,文件子域2] ...]
;将 INF 文件拷贝到系统的 Windows\Inf 目录下
[CopyINF = xxx.inf[,xxx.inf]...] ;指令参数为多个 INF 文件名
;注册表操作
;AddReg: 添加; DelReg: 删除
;BitReg: 修改已经存在的类型为 REG_BINARY 的注册表键值
[AddReg = 子域名[,子域名2]...]
[DelReg = 子域名[,子域名2]...]
[BitReg = 子域名[,子域名2]...]
;包含
[Include = xxx.inf[,xxx.inf]...] ;指令参数为多个 INF 文件名
[ Needs = INF 文件域名1 [,INF 文件域名2]...] ;参数为 Include 指令中 INF 文件域名
;新建、删除“开始”菜单项
[ProfileItems = 子域名1[,子域名2]...]
;INI文件操作
[UpdateInis = 子域名[,子域名2]...]
[UpdateIniFileds = 子域名[,子域名2]...]
[Ini2Reg = 子域名[,子域名2]...]
;DLL 操作
[RegisterDlls = 子域名[,子域名2]...]
[UnregisterDlls = 子域名[,子域名2]...] ...
ProfileItems 指令用来创建、删除“开始”菜单项。指令的详细讲解此处省略，读者应能通过下面的实验领会其使用方法。
实验：在“开始”菜单中新建快捷方式
[DefaultInstall]
ProfileItems = NewDir ,NewItem
; Start\Program 下新建目录 NewDir
[NewDir]
Name = "NewDir",4
CmdLine = 11,,,
;NewItem 子目录下新建快捷方式
[NewItem]
;首先是名称
;在没有其他参数的情况下是新建
;删除需要设置参数2: Name = “计算器”, 2
Name = "计算器.lnk"
```

```
;执行命令，打开 system32 目录下的计算器程序
CmdLine = 11,, calc.exe
;快捷方式的创建位置
SubDir = "NewDir"
;cmd 程序运行时的起始位置
WorkingDir = 11

;ToolTip 信息
InfoTip = "@%\shell32.dll,-22531"
;显示的名称；如果不设置，将显示 Name 所设置的名称
DisplayResource = "%\shell32.dll",22019
```

2.[DefaultInstall.Services]子域

这是一个可选的服务子域，通过调用 AddService 或 DelService 指令进行服务的注册或删除操作。这两个指令上面已经详细讲过，这里省略。指令格式如下：

```
[DefaultInstall.Services]
;AddService 为必选命令；可以有多个 AddService 指令
;它用来在上面的注册表键下添加新的服务子键
AddService = ...
;可选的删除指令
[DelService = ...]
[Include = filename.inf[,filename2.inf]...]
[Needs = inf-section-name[,inf-section-name]...]
```

X. 附录

X.I AddService指令Flags

标识宏	值	含义
SPSVCINST_TAGTOFRONT	0x00000001	Move the named service's tag to the front of its group order list, thereby ensuring that it is loaded first within that group (unless a subsequently installed device with this INF specification displaces it). INF files that install exclusively PnP devices and devices with WDM drivers should not set this flag.
		Assign the named service as the PnP

SPSVCINST_ASSOCSERVICE	0x00000002	function driver (or legacy driver) for the device being installed by this INF file. Do not specify this flag when installing filter drivers or other driver components not directly associated with a device. Set this flag for only one driver for each INF DDInstall.Services section.
SPSVCINST_NOCLOBBER_DISPLAYNAME	0x00000008	Do not overwrite the given service's (optional) friendly name if this service already exists in the system.
SPSVCINST_NOCLOBBER_STARTTYPE	0x00000010	Do not overwrite the given service's start type if this named service already exists in the system.
SPSVCINST_NOCLOBBER_ERRORCONTROL	0x00000020	Do not overwrite the given service's error-control value if this named service already exists in the system.
SPSVCINST_NOCLOBBER_LOADORDERGROUP	0x00000040	Do not overwrite the given service's load-order-group value if this named service already exists in the system. INF files that install exclusively PnP devices and devices with WDM drivers should not set this

		flag.
SPSVCINST_NOCLOBBER_DEPENDENCIES	0x00000080	Do not overwrite the given service's dependencies list if this named service already exists in the system. INF files that install exclusively PnP devices and devices with WDM drivers should not set this flag.
SPSVCINST_NOCLOBBER_DESCRIPTION	0x00000100	Do not overwrite the given service's (optional) description if this service already exists in the system.
SPSVCINST_CLOBBER_SECURITY	0x00000400	(Windows XP and later versions of Windows) Overwrite the security settings for the service if this service already exists in the system.
SPSVCSINST_STARTSERVICE	0x00000800	(Windows Vista and later versions of Windows) Start the service after the service is installed. This flag cannot be used to start a service that implements a Plug and Play (PnP) function driver or filter driver for a device. Otherwise, this flag can be used to start a user-mode or kernel-mode service that is managed by the

		Service Control Manager (SCM).
SPSVCINST_NOCLOBBER_REQUIREDPRIVILEGES	0x00001000	(Windows 7 and later versions of Windows) Do not overwrite the privileges for the given service if this service already exists in the system.

X.II 预定义的目录ID

系统目录ID	路径	Shell目录ID	路径(SetupAPI.h/Shlobj.h)
01	SourceDrive:\pathname(当前的安装文件所在目录)	16406	All Users\Start Menu
10	Windows 文件夹，等同于：%SystemRoot%	16407	All Users\Start Menu\Programs
11	系统文件夹，等同于：%SystemRoot%\System32	16408	All Users\Start Menu\Programs\Startup
12	Drivers 文件夹，等同于：%SystemRoot%\System32\drivers	16409	All Users\Desktop
17	INF 文件夹，等同于：%SystemRoot%\Inf	16415	All Users\Favorites
18	帮助文件夹	16419	All Users\Application Data
20	字体文件夹	16422	Program Files
21	Viwers 文件夹	16425	%SystemRoot%\system32(WOW64(Windows on Windows)模式下有效)
23	色彩目录(ICM)(打印机驱动不适用)	16426	Program Files(WOW64模式下有效)
24	系统盘根目录。操作系统安装在哪里目录 ID 即代表哪个盘，如 C:\	16427	Program Files\Cpmmon
25	共享目录	16428	Program Files\Cpmmon(WOW64模式下有效)
30	启动盘根目录。也就是引导分区所在的那个磁盘，或者说是物理上的第一个分区盘。 若不出意外，总是等于 C:\。他不一定等于目录 ID24 所代表的目录，另	16429	All Users\Templates

	外，系统可以安装在非启动分区上		
50	系统文件夹(!=11)，等同于：%SystemRoot%\system	16430	All Users\Documents
51	Spool 目录(打印机驱动不适用)		
52	Spool 驱动目录(打印机驱动不适用)		
53	用户目录		
54	系统引导文件 Ntldr.exe 和 Osloader.exe 所在的目录。一般情况下，等同于 30 所表示的目录		
55	Print processor 目录(打印机驱动不适用)		
-1	绝对路径，即:不使用任何目录 ID		