<http://www.cnblogs.com/Cindy_weiwei/archive/2009/05/18/1459542.html>

[**【海洋女神原创】一个完整的Installshield安装程序实例—艾泽拉斯之海洋女神出品(一）---基本设置一**](http://www.cnblogs.com/Cindy_weiwei/archive/2009/05/18/1459542.html)

**前言**

Installshield可以说是最好的做安装程序的商业软件之一，不过因为功能的太过于强大，以至于上手和精通都不是容易的事情，之前都是用Installshield的Project Assistant对付过去的，这次做这个安装程序，为了实现一些功能，必须写代码，国内外现成的资料很少，而且很多都语焉不详，自己反复啃了多次，对比Installshiel自带的help，才明白资料所表达的意思。这个安装程序虽然比较简陋，在行家眼里可能是小菜一碟，但是也花了笔者一个星期的时间，阅读了很多资料，啃了好几天英文help，集成了很多先驱者的经验，也费了自己不少心血做成的，对每一段代码的用处、每一个用到的函数都进行了详细的说明，因此转载时请务必保留转载出处和由艾泽拉斯之海洋女神出品的字样；如需刊登，请与作者联系。

在此要感谢吞硬币的小猪，天下晓明，余满青，海洋C++乐园（此海洋不是彼海洋）等大虾在互联网上的无私奉献，他们的贴子和博客给了我很大启示。

因为本人是做java出身的，因此对这种类C++语言还是第一次接触，有理解不当之处，请朋友们指正。欢迎Email至[little\_fairycat@126.com](mailto:little_fairycat@126.com)

**正文**

需求：公司做了一个软件产品，

1.         该软件运行需要JDK环境（不是JRE，因为该软件要向windows注册一个服务，用到了JavaService，JDK才支持这个功能；不过这里侧重于判断是否安装了某软件是否安装，而不是纠缠于该装JDK还是该装JRE）；

2.         由于是Server-Client形式的，需要允许用户选择安装组件，比如A机只装Server端，B机只装Client端；

3.         文档不打包在安装程序里，直接存放在光盘文件夹下方便用户查看，同时允许用户指定是否安装文档到计算机上（为什么这样做，后面说明详细原因）；

4.         该软件会以授权形式发放给用户，不同的用户，软件本身可能相同，而不同的只是授权文件和一些配置，因此希望授权文件和配置文件不打包在安装程序里，而直接存放在光盘里，以减少可能的重复打包安装程序的劳动；

5.         在安装完毕后，希望能自启动程序（因为该软件需要在安装完毕后启动一个程序，该程序实现向Windows注册服务的功能，该程序最好由安装程序启动，而不是由客户手动启动）。

6.         希望有反安装程序

本文提到的“外部”指不打包在安装程序里的，与安装程序一起存放在光盘里的一些文件夹，这些文件夹包含了安装中所需要的文件，同时也可能有其他用途，因此不适合直接压缩打包在安装程序里。

该实例实现了如下功能：

1.         显示软件许可协议

2.         判断是否安装了本软件所需要的先决软件JKD1.6.0\_04，如无，则启动外部安装程序进行安装（同样原理可以用来判断是否安装了其他软件，只要该软件在注册表中有键值）

3.         安装允许用户选择需要安装的组件

4.         用户的输入信息、所选安装路径、所选安装组件将显示在安装界面上（Installshield虽然自带了此界面，但是默认是显示为空的，需要写脚本来显示信息）

5.         根据用户选择的组件，在开始菜单显示程序的快捷方式（同样适用于桌面快捷方式，后面做详细说明）

6.         根据用户选择的组件，从外部文件夹拷贝相应的文件到安装目标路径的文件夹中

7.         根据从外部拷贝进来的文件，创建快捷方式（这里主要是拷贝文档，并在开始菜单中创建快捷方式）

8.         在安装结束时，显示readme.txt文件

9.         在安装结束后，启动指定的程序

10.     完美卸载

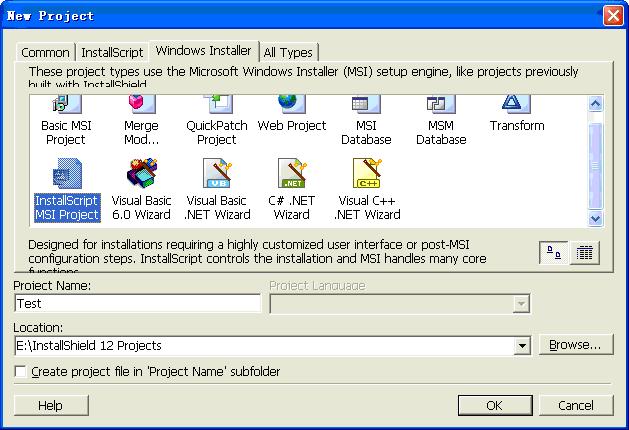
笔者所用的环境为Installshield 12 Premier Edition，Windows XP with SP2, 该环境下建立的工程可以直接使用在Installshield 2008 Premier Edition下，Installshield 2008在打开Installshield 12所建的工程时会提示你是否需要进行Upgrade，确认即可，软件会自动为你进行升级，很方便。

下面我们一步一步来建立一个基本的工程，并且使用脚本来完善和丰富所需功能

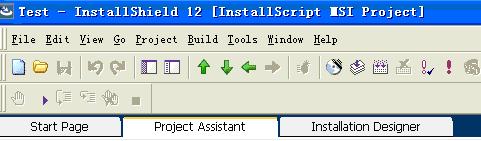
**第一部分：基本设置**

1.         打开Installshield 12 Premier Edition，新建一个Installscript MSI Project，这种被称之为半脚本程序，因为兼具Basic Project基本类型和Installscript Project全脚本类型两者的优点，我比较喜欢用。像我这样需求的，既要用到Wizard的便利，又想写一点脚本来实现一点自定义操作的，就比较适合用这种类型啦。

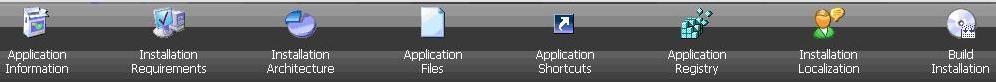
选择类型为Windows Installer | InstallScript MSI Projcet，输入工程名，指定工程所在的文件夹。



2.         界面会切换到Project Assistant，我们先从这里开始把工程的基本组件和基本文件建立好。



3.      在Project Assistant界面的底部，会有一个引导动作条，在建立该工程的基本结构和文件时，我们都将在此界面进行操作，下文都将以“引导条”来指代这个引导动作条。



4.         点击引导条上的Application Information

这里输入：

公司名，公司名将会出现在Setup.exe的注解中

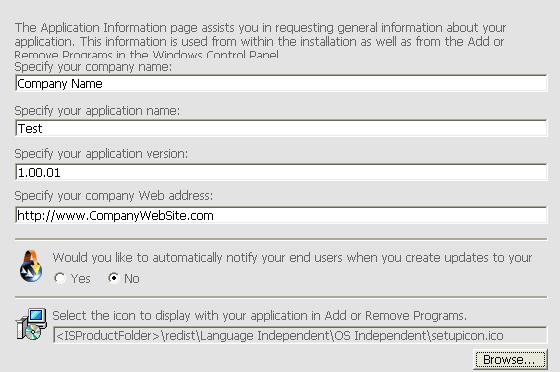
软件名，将会出现在安装过程的左上角标题栏上

版本号，没看到在哪，不过自己比较方便地知道自己在编译哪个版本的软件

公司网址，没看到在哪，而且如果该公司没有网址呢？而且这里有点bug，好像默认的值总是会报一个String\_ID1为空的错误，自己输入一个网址就不会报错。

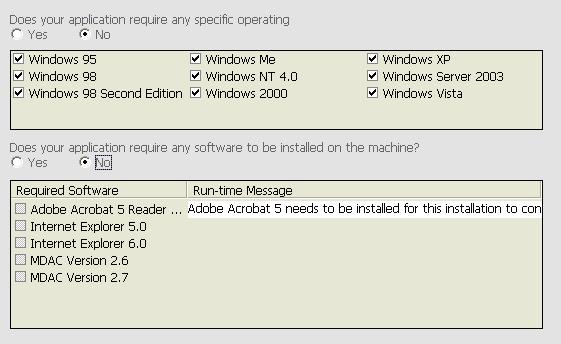
是否在你创建了更新时自动通知最终用户，没用过，我都选了No。

选择一个图标，这个图标会出现在“添加或删除程序”里，我一般用默认的，当然你可以替换成自己想要的图标。



5.         点击引导条上的Installation Requirement

这里选择对操作系统和一些软件的需求。根据自己需要来选择是否要求操作系统的版本，已经是否要求安装了某些软件。



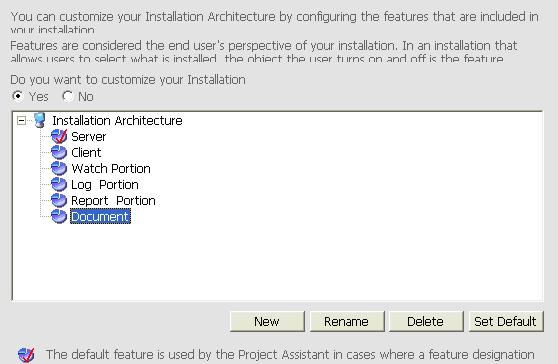
6.         点击引导条上的Installation Architecture

这是个十分有用的设置，对于本文所用的分布式软件来说非常合适，分布式软件的每个组件可以设置为一个Feature，用户可以自由选择安装某些功能。

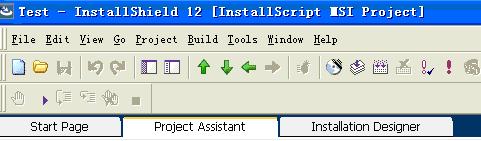
将选项Do you want to customize your Installation选择为Yes。

点击选中根节点Installation Architecture，点击New创建新的Feature，可以为每个Feature指定新名称。

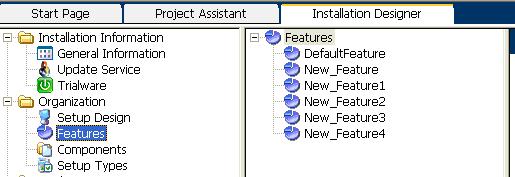
还可以在Feature下创建子Feature，比如如果文档Feature下包括软件本身文档，和软件所需的运行环境的文档，那么可以创建两个子Feature，分别包含两种文档，用户在安装时就可以选择安装部分或者全部文档了。这里我们没有用到子Feature，用途和普通Feature一样。



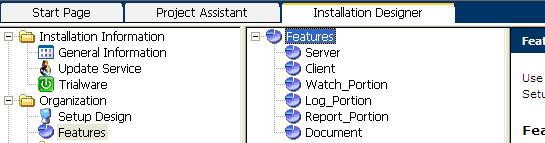
这里，建立好所有Feature后，我们将切换到Installation Designer做一个设置



找到Installation Designer页面上左边导航树Organization | Features分支，你会看到这里Features都显示为原始的名称，而非我们改过的名字，因为Feature有Name和Display Name两种名称，我们刚才改的不过是Display Name，为了便于查看和使用，我们在这里把Name也改一下



注意Name不可以有空格，可以使用下划线



继续切换回Project Assistant

## [【海洋女神原创】一个完整的Installshield安装程序实例—艾泽拉斯之海洋女神出品(二) --基本设置二](http://www.cnblogs.com/Cindy_weiwei/archive/2009/05/19/1460230.html)

上一篇：[一个完整的安装程序实例—艾泽拉斯之海洋女神出品(一）---基本设置一](http://www.cnblogs.com/Cindy_weiwei/archive/2009/05/18/1459542.html)  
7. 点击引导条上的Application Files

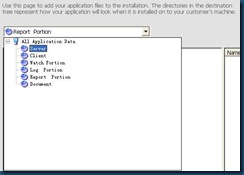
我们将在这里对安装路径进行微调，并且为每个Feature指定需要安装的文件

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image002_2.jpg)

这里我不想使用Program Files | Company Name | Product Name这个路径，我想使用Program Files | Product Name，我直接点击选中My Product Name[INSTALLDIR]拖动到ProgramFileFolder下，还可以直接将My Product Name 改成自己想要的文件夹名字

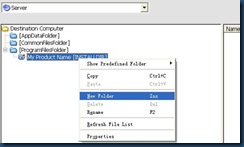
[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image004_2.jpg)

接下来，为每个Feature指定要安装的文件。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image006_2.jpg)

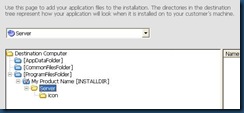
打开这个下拉列表，所有的Feature都在这里，按顺序来给每一个Feature建立文件夹，并且导入所需的文件。

选择第一个Feature， 即Server，点击My Product Name[INSTALLDIR]节点，右键点击，在菜单上选择New Folder来创建一个文件夹。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image008_2.jpg)

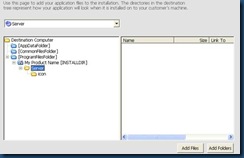
创建一个Server文件夹，这个文件夹将用来存放该组件需要的一些文件。

再在Server文件夹下创建一个icon文件夹，存放该组件所用的图标。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image010_2.jpg)

然后为该Feature添加安装时该Feature要安装的文件。

这里我们建立的icon文件夹是用来存放这个feature在后面要建立快捷方式时使用的图标的。为这个icon文件夹添加相应的图标文件，并且记住图标文件的来源文件夹，后面设置快捷方式的时候要用。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image012_2.jpg)

点击选中要添加文件的文件夹，然后点击右下角的Add Files，然后添加文件

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image014_2.jpg)

接下来我们为Feature添加文件夹，如果这个文件夹中的全部文件都为这个Feature所需。添加文件夹的好处在于只要文件夹位置和名称不变，那么文件夹里面的文件都是动态加载的，有多少加载多少，不用考虑文件名的改动带来的影响。

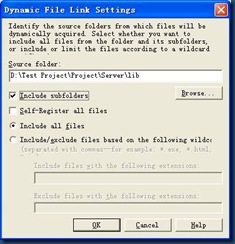
点击选中要添加文件夹的文件夹，然后点击右下角的Add Folders，然后添加文件夹。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image016_2.jpg)

选中文件夹，点击确定。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image018_2.jpg)

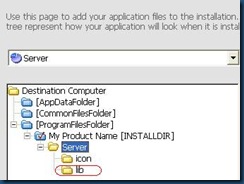
会询问你是否要使用动态文件链接，我都选择确定，好处就在于我刚才上面所述。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image020_2.jpg)

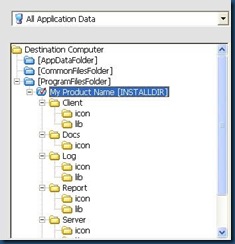
显示了源文件夹，如果这个文件夹下有子文件夹，并且也需要一并添加进来的话，务必钩选Include subfolders选项。

这里还允许做一些简单设置来包含或者排除一些特定文件，支持通配符。

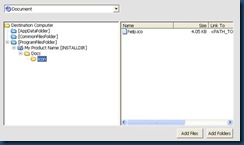
点击OK确定加入文件夹。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image022_2.jpg)

如法炮制为每个Feature建立文件夹，并且添加文件，最后效果如图所示

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image024_2.jpg)

Document这个Feature，除了文件所用的图标外，什么都不要添加，后面我们将用安装时实时拷贝的方式来拷贝文档进来。

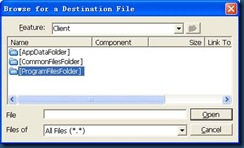
[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image026_2.jpg)

8. 接下来我们为可执行文件创建快捷方式。

点击引导条上的Application Shortcuts

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image028_2.jpg)

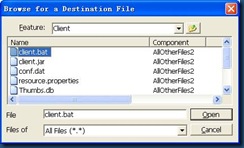
点击New新建一个快捷方式

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image030_2.jpg)

选择一个要建立快捷方式的Feature。

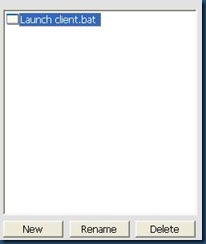
如果要建立快捷方式的程序为非.exe形式，请把Files Of选择选为All Files(\*.\*)格式。

我们的程序安装目标路径设置在Program Files下，因此双击[ProgramFilesFolder]打开，层层点击进入。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image032_2.jpg)

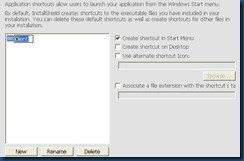
我们这里要为client.bat建立一个快捷方式，因为这个是启动用的批处理文件。

Installshield可以自动监测到.exe文件的存在，自动生成快捷方式，用户只需要做一些适当修改即可。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image034_2.jpg)

新建的快捷方式将出现在这里，名字不好听，样子也不好看，我们将为它改一个名字，并且换一个图标。

选中快捷方式，点击Rename，并且为这个快捷方式改一个适当的名字。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image036_2.jpg)

注意右边的几个选项。

Create shortcut in Start Menu，将在开始菜单里创建一个快捷方式。

Create shortcut on Desktop，将在桌面上创建一个快捷方式。

Use alternate shortcut Icon，替换快捷方式的图标

Associate a file extension with the shortcut’s target，没用过，不知道什么意思。

我们在这里将只创建开始菜单的快捷方式，因此钩选第一项。

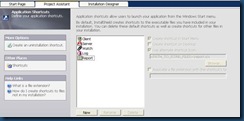
钩选第三项，并且点击Browse来浏览图标。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image038_2.jpg)

请回想刚才在为Feature添加文件的时候，每个feature都添加了对应的icon。这里，请把浏览的文件夹设定为刚才添加icons所用的文件夹，通俗的说，就是你刚才从哪儿添加一个图标进feature的，现在还是从哪儿添加的这个图标。

其实这一点我是一直很费解的，当初不知道要这么选择图标，随便从外面一个任意文件夹里添加了一个图标，以至于打包后死活找不到图标，后来经过试验才知道这个被选中的图标文件要拷贝进来，打包进安装文件才可以。这一点上不能不提一下visual studio，这个工具做安装程序虽然功能一般，但是思想还是不错的，当它的组件指定拷贝了图标文件后，在建立快捷方式时，快捷方式使用的图标是指向虚拟的安装目标路径下的图标文件的，而不是指定到这个实实在在的源文件夹。这一点差别就体现出了思想上的差异。

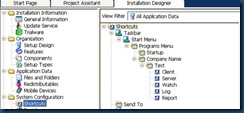
如法炮制为每个Feature指定快捷方式，Document除外，因为我们在这个feature里除了图标文件外什么都没有添加。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image040_2.jpg)

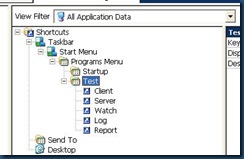
至此我们为每个可执行程序添加了开始菜单下的快捷方式。

我们再切换去Installation Designer，找到System Configuration | Shortcuts。

看到快捷方式在开始菜单中是以　公司名　｜　软件名　｜　快捷方式　　　这种形式存在的。事实上我是不喜欢这种形式了，想想点开一层还有一层，不如直接了当来得干脆，因此做一些修改。

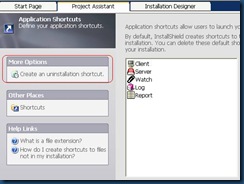
[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image042_2.jpg)

这里我改成了如下设置

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image044_2.jpg)

不要告诉我你不会改，直接拖动Test文件夹往Program Menu（即开始菜单下的那个“所有程序”）下一塞即可，然后删除掉多余的Company Name文件夹。

9. 可能刚才在Project Assistant界面有人已经注意到了左边栏上More Options下Create an uninstallation shortcut这个诱人的字样了。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image046_2.jpg)

可是我要告诉你，如果你选择了这种方式建立卸载快捷方式的话，你会很沮丧地发现：

a) 似乎只有在安装某个feature的时候这个卸载快捷方式才会出现（当然，就是那个default feature，这种要命的feature形式决定了每个文件或者快捷方式都必须明确地归属到某个feature下），因此，当你的客户只选择了其他feature安装时，这个卸载方式不会出现，而他必须去“添加或卸载程序”里面去卸载

b) 如果你写脚本使得安装时会拷贝一些外部文件进来，那么这些文件在这种卸载方式下是删除不掉的。（如果你确实想保存这些文件，你可以在脚本里设置它们属性为permanent，这个属性可以保证什么卸载方式都不能删除你的这些文件）。

所以这里我们忽视这个卸载快捷方式的存在，而将在后面采用脚本形式实现完美卸载。

10. 点击引导条上的Application Registry

向注册表写键和键值，由于本工程不需要，忽略之。有需要的朋友可以查阅相关资料，不难。

11. 点击引导条上的Installation Location

这个是用来设置安装包的语言的，选择了多个语言后，用户可以在安装界面开始的时候选择安装时所用的语言；不过作为一个公司产品来说，这么偷懒，客户的印象是要打折扣的，所以还是选个单语言吧，该什么语言的安装包就什么语言的安装包，各归各。

不过你又会沮丧地发现，如果要选择一种其他语言作为Default Language，好像又报错了。

这个问题当时折腾了我一个星期（当然那时候才接触Installshield，还一窍不通），最后问了技术支持才得以解决。

切换去Installation Designer，找到Installation Information | General Information，看到String Tables下面是什么？对，所有你选的语言都列出来了，选中你要的语言，右键，选择Make Default，OK，再切换回Project Assistant去把所有不要的语言统统去掉钩选即可。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image048_2.jpg)

看到此处，已经变成了English为默认语言了。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/e68d4c8b8326_D397/clip_image050_2.jpg)

12. 点击引导条上的Build Installation。

打包安装盘的设置，本人从来不用这个选项，都用工具条上的Release Wizard。

至此，第一部分基本完成。如果是一些没有特别要求的安装包，这部分讲解的内容足够可以做一个基本的安装包了

### 第二部分：脚本编程

在开始进行编程前，我们先明确一下我们要用编程来弥补前面设置的哪些功能的不足

1. 显示软件许可协议

2. 判断是否安装了本软件所需要的先决软件JKD1.6.0\_04，如无，则启动外部安装程序进行安装（同样原理可以用来判断是否安装了其他软件，只要该软件在注册表中有键值）

3. 用户的输入信息、所选安装路径、所选安装组件将显示在安装界面上（Installshield虽然自带了此界面，但是默认是显示为空的，需要写脚本来显示信息）

4. 根据用户选择的组件，从外部文件夹拷贝相应的文件到安装目标路径的文件夹中

5. 根据从外部拷贝进来的文件，创建快捷方式（这里主要是拷贝文档，并在开始菜单中创建快捷方式）

6. 在安装结束时，显示readme.txt文件

7. 在安装结束后，启动指定的程序

8. 完美卸载

脚本编程这部分都将在Installer Designer这个界面进行。后面不再赘述。

Installshield大小写敏感，因此请严格按照示例上所写的大小写规则来书写。例：字符串变量STRING和string都支持，但是String不支持。

#### 1. 显示软件许可协议

1. 添加许可协议文本

在左边导航树上找到Behavior and Logic | Support Files/Billboards选项。这个选项允许用户添加一些在安装过程中需要用到的文件。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image002_2.jpg)

中间的导航栏会显示对应的选项

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image004_2.jpg)

在Support Files分支下，会显示一个Language Independent和所有你所选择的语言类型。 Language Independent意为，如果你在这里分支下做了设置，那么无论选择用何种语言安装，这个设置都会生效；而各个语言类型意为，如果你在某语言下做了设置，那么这个设置只有在选择了用这种语言安装的时候才会生效。

点击Language Independent，这次我们将在这个分支下进行试验。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image006_2.jpg)

在右边的Files栏中右键点击，在弹出菜单上选择Insert Files选项。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image008_2.jpg)

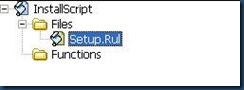
选择事先撰写好的许可协议的文本文件，插入到Files栏中。

许可协议允许两种文本格式：txt和rtf格式，此处我们采用 txt格式。

2. 然后切换到Behavior and Logic | InstallScript选项，

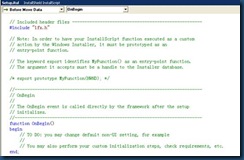
[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image010_2.jpg)

3. 中间的导航栏Files下有一个默认的Rul文件Setup.Rul，我们这个工程的全部installscript代码都将写在这个默认文件里

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image012_2.jpg)

4. 点击选中Setup.Rul节点，右边会显示该文件的可编程面板。

5. 许可协议应该在一开始运行安装程序的时候就显示，也就是在拷贝数据前。请在第一个下拉框中选择Before Move Data选项，然后在第二个下拉框中选择OnBegin选项（不要因为默认显示的是这两个选项，而不做这个打开下拉列表进行选择的动作，否则软件检测不到你选择了选项，无法自动添加代码），则编程界面上会自动添加一些代码如下图所示。当然，如果你手动敲代码上去也是可以的。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image014_2.jpg)

6. 我们将在function OnBegin()的函数体里面写代码来显示刚才添加的许可协议文本的内容，直接把下面的代码拷贝到OnBegin()函数的begin和end;之间就可以了

Disable (BACKBUTTON);

if(!MAINTENANCE)then

SdLicense2 ("License ", "", "", SUPPORTDIR ^ "2.txt", FALSE);

endif;.

7. 代码解释

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Disable (BACKBUTTON);

将“上一步”按键设置为不可用。安装程序在一开始的时候会有一个默认的开始界面，第二步才显示许可协议，一般来说没必要回退回去看这个什么都没有的开始界面，因此将回退按键设置为不可用

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(!MAINTENANCE)then

endif;

这一个条件用来判断安装程序处于何种状态，安装、修复、重新安装或卸载状态，后三者都属于MAINTENANCE状态，因此判断只有在正常安装的状态才显示许可协议

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SdLicense2 ("License ", "", "", SUPPORTDIR ^ "2.txt", FALSE);

这个函数用于在界面上显示所用的许可协议。Help里对该函数的构造函数如下

SdLicense2 ( szTitle, szOpt1, szOpt2, szLicenseFile, bLicenseAccepted );

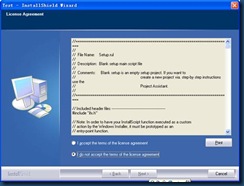
参数一：szTitle，显示在界面左上角的标题，如果填写空字符串””，则显示为默认值”License Agreement”。

参数二：szOpt1，我们常见许可协议界面上会有两个选项，一个是“同意”，一个是“不同意”，szOpt1和szOpt2就是这两个选项，如果填写空字符串，则会显示为默认值"I accept the terms of the license agreement"和"I do not accept the terms of the license agreement"。

参数三：szOpt2，见参数二的说明

参数四：szLicenseFile，指定需要显示的文档，包含路径和带扩展名的文档名。我们刚才把许可协议文本放在supportfile选项下了，这个路径在Installshield里有专门的静态变量来指明，即SUPPORTDIR，然后再添加上带扩展名的文档名，这里是2.txt。静态变量路径和引号引起来的路径之间用^符号来连接。

参数四：bLicenseAccepted，布尔型变量，TRUE状态，则在许可协议界面上默认选中的是那个“同意”的选项；不过好像一般更常见的是默认选中为“不同意”的选项，因此这里可以填入FALSE。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image016_2.jpg)

这是许可协议的界面。当用户选择了I accept the terms of the license agreement这个选项后，Next按键可用，安装程序可以继续。（请忽略这里显示的许可协议内容…网上有很多软件许可协议的范本供下载...）

小结：至此，许可协议就添加完毕，在安装执行的时候，用户就可以看到许可协议显示在界面上，并且只有选择了“同意”选项后，安装程序才会往下执行。

显示许可协议的函数一共有三个SdLicense，SdLicenseRtf和SdLicense2，参数略有不同，显示的界面也略有不同，用户可以根据喜好来选择。目前我常用的就是SdLicense2这个函数，显示的界面符合大多数目前流行的安装界面的习惯。

#### 2. 判断是否安装了本软件所需要的先决软件或运行环境

1. 代码还是在OnBegin()函数体内实现，直接把下面的代码拷贝到OnBegin()函数的begin和end;之间就可以了

RegDBSetDefaultRoot(HKEY\_LOCAL\_MACHINE);

if (RegDBKeyExist ("SOFTWARE\\JavaSoft\\Java Development Kit\\1.6.0\_04") < 0) then

LaunchAppAndWait (SRCDISK^"jdk\\jdk-6u4-windows-i586-p.exe","", LAAW\_OPTION\_WAIT);

endif;

2. 代码解释

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RegDBSetDefaultRoot(HKEY\_LOCAL\_MACHINE);

设置一下默认的注册表键值根节点为HKEY\_LOCAL\_MACHINE。

打开注册表可以看到“我的电脑”下的根节点有HKEY\_CLASSES\_ROOT， HKEY\_CURRENT\_USER，HKEY\_LOCAL\_MACHINE等。我们这次要寻找的JDK软件的注册表键值在HKEY\_LOCAL\_MACHINE下，因此要把根键设置为HKEY\_LOCAL\_MACHINE。

表告诉我你不知道怎么看注册表，开始-〉运行-〉输入命令regedit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RegDBKeyExist ("SOFTWARE\\JavaSoft\\Java Development Kit\\1.6.0\_04") < 0)

判断是否存在键值SOFTWARE\\JavaSoft\\Java Development Kit\\1.6.0\_04，这个是JDK1.6.0\_04安装时向注册表写入的值；

RegDBKeyExist( szSubKey );如果存在键值则返回1，否则返回小于0的随机数字。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LaunchAppAndWait (SRCDISK^"jdk\\jdk-6u4-windows-i586-p.exe","", LAAW\_OPTION\_WAIT);

当上面判断了没有安装JDK1.6.0\_04这个软件时，则启动光盘里jdk文件夹下的jdk-6u4-windows-i586-p.exe安装程序来安装。

这个函数在help里是这样叙述的：

LaunchAppAndWait ( szProgram, szCmdLine, nOptions );

参数一：szProgram，即要启动的程序。这里我们写入的参数是SRCDISK^"jdk\\jdk-6u4-windows-i586-p.exe"， SRCDISK指源盘，安装程序所在的盘，光盘和硬盘都可以。"jdk\\jdk-6u4-windows-i586-p.exe"源盘下jdk文件夹下的jdk-6u4-windows-i586-p.exe安装程序。

参数二：szCmdLine，如果要启动的程序需要从命令行读入参数来启动，那么在这里写入对应的参数值；我们这里不需要，因此输入空字符串””。

参数三：nOptions，静态变量，不同的静态变量会得到不同的执行结果，比如无等待安装，静默安装，鼠标外形改变等等。详情请参阅Installshield自带的Help。这里我们用LAAW\_OPTION\_WAIT，即当JDK安装结束后（无论是正常安装了，还是用户点击取消了安装），安装程序才往下继续。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image018_2.jpg)

这里可以看到，当点击了同意许可协议的时候，安装程序会自动检测是否安装了JDK，如果没有安装，则弹出安装界面。

这里在函数体里面，没有对找不到JDK安装程序，以及安装出错等情况做判断。如果用户有需要，可以添加一个消息框，提示在找不到安装程序或者安装出错的情况下，用户可以手动地安装需要的软件。代码可以改写为

RegDBSetDefaultRoot(HKEY\_LOCAL\_MACHINE);

if (RegDBKeyExist ("SOFTWARE\\JavaSoft\\Java Development Kit\\1.6.0\_04") < 0) then

if(LaunchAppAndWait (SRCDISK^"jdk\\jdk-6u4-windows-i586-p.exe","", LAAW\_OPTION\_WAIT)<0)then

MessageBox ("You haven't installed JDK 1.6.0\_04 yet! ", INFORMATION);

endif;

endif;

小结：至此，判断运行所需软件的功能结束，用户可以自己试验一下判断多个软件。用法就是重复上述代码功能，仍在OnBegin()函数体内执行。

#### 3. 用户的输入信息、所选安装路径、所选安装组件显示在安装界面上

Installshield是自带这个界面的，在安装过程中用户可以看到这个界面，但是这个界面上的信息是空的，这一点让人很是疑惑，怀疑是Installshield的bug。因此我们不得不手动地实现这个功能。

1. 这个功能需要在OnFirstUIBefore()函数体中实现，选择Before Move Data | OnFirstUIBefore选项

[clip_image020](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image020_2.jpg)

2. 选择了这个选项后，软件会自动在编程界面生成大量代码，如图所示，这里的每一个Dlg\_SdXXXX都对应着一个界面，例如Dlg\_SdWelcome就是对应着最初开始的欢迎界面。如果开发人员对这些很熟悉，可以在这里对每一个界面编程设置。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image022_2.jpg)

3. 找到Dlg\_SdStartCopy这个界面选项，我们将在这里对已有的代码进行改动，使之显示用户输入的用户信息、所选安装路径和组件等信息

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image024_2.jpg)

4. 首先定义所需变量。

在begin前定义6个feature的名字和两个NUMBER类型的变量，即蓝色字串。之前在第一部分我们定义了6个可用的feature，这里就要对这6个feature进行一些判断。

在begin字样后对这6个feature赋值，所赋的值就是我们在第一部分定义的feature的名字(Name, not Display Name)。

//---------------------------------------------------------------------------

function OnFirstUIBefore()

NUMBER nResult, nSetupType, nvSize, nUser;

STRING szTitle, szMsg, szQuestion, svName, svCompany, szFile;

STRING szLicenseFile;

LIST list, listStartCopy;

BOOL bCustom;

STRING szFeatureName1;

STRING szFeatureName2;

STRING szFeatureName3;

STRING szFeatureName4;

STRING szFeatureName5;

STRING szFeatureName6;

NUMBER bvOpt1,bvOpt2;

begin

// TO DO: if you want to enable background, window title, and caption bar title

// SetTitle( @PRODUCT\_NAME, 24, WHITE );

// SetTitle( @PRODUCT\_NAME, 0, BACKGROUNDCAPTION );

// Enable( FULLWINDOWMODE );

// Enable( BACKGROUND );

// SetColor(BACKGROUND,RGB (0, 128, 128));

szFeatureName1 ="Server";

szFeatureName2 ="Client";

szFeatureName3 ="Watch\_Portion";

szFeatureName4 ="Log\_Portion";

szFeatureName5 ="Report\_Portion";

szFeatureName6 ="Document";

5. 在Dlg\_SdStartCopy的listStartCopy = ListCreate( STRINGLIST ); 和ListDestroy(listStartCopy);之间的nResult = SdStartCopy( szTitle, szMsg, listStartCopy );之前加入如下代码。

ListAddString(listStartCopy,"Customer Information:",AFTER);

ListAddString(listStartCopy,"User Name: " + svName,AFTER);

ListAddString(listStartCopy,"Company Name: " + svCompany,AFTER);

ListAddString(listStartCopy,"Destination Location: " + INSTALLDIR,AFTER);

switch (nSetupType)

case TYPICAL : ListAddString(listStartCopy,"Setup Type: Typical",AFTER);

case COMPACT: ListAddString(listStartCopy,"Setup Type: Compact",AFTER);

case CUSTOM: ListAddString(listStartCopy,"Setup Type: Custom",AFTER);

endswitch;

ListAddString(listStartCopy," ",AFTER);

ListAddString(listStartCopy,"The Selected Feature:",AFTER);

if (FeatureIsItemSelected(MEDIA, szFeatureName1)=1) then

ListAddString(listStartCopy," "+szFeatureName1,AFTER);

endif;

if (FeatureIsItemSelected(MEDIA, szFeatureName2)=1) then

ListAddString(listStartCopy," "+szFeatureName2,AFTER);

endif;

if (FeatureIsItemSelected(MEDIA, szFeatureName3)=1) then

ListAddString(listStartCopy," "+szFeatureName3,AFTER);

endif;

if (FeatureIsItemSelected(MEDIA, szFeatureName4)=1) then

ListAddString(listStartCopy," "+szFeatureName4,AFTER);

endif;

if (FeatureIsItemSelected(MEDIA, szFeatureName5)=1) then

ListAddString(listStartCopy," "+szFeatureName5,AFTER);

endif;

if (FeatureIsItemSelected(MEDIA, szFeatureName6)=1) then

ListAddString(listStartCopy," "+szFeatureName6,AFTER);

endif;

6. 代码解释

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ListAddString(listStartCopy,"XXXXXX",AFTER);

把要显示的信息添加到list里去，这个list的内容稍后会添加到界面上进行显示。

Help里对这个函数是这样描述的：ListAddString ( listID, szString, nPlacementFlag );

参数一：listID，需要用户事先创建一个list，这里我们看到listStartCopy = ListCreate( STRINGLIST );这句话，即创建了一个叫listStartCopy的list

参数二：szString，要添加的字符串

参数三：nPlacementFlag，如果设置为AFTER，则顺序添加；如果为BEFORE，则逆序添加，即新添加的内容会放在前面显示。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

switch (nSetupType)

case TYPICAL : ListAddString(listStartCopy,"Setup Type: Typical",AFTER);

case COMPACT: ListAddString(listStartCopy,"Setup Type: Compact",AFTER);

case CUSTOM: ListAddString(listStartCopy,"Setup Type: Custom",AFTER);

endswitch;

这是根据用户选择的安装类型来显示安装类型信息。安装类型分三种：TYPICAL，COMPACT和CUSTOM。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (FeatureIsItemSelected(MEDIA, szFeatureName1)=1) then

ListAddString(listStartCopy," "+szFeatureName1,AFTER);

endif;

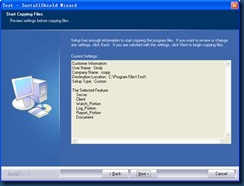
这里的FeatureIsItemSelected(MEDIA, szFeatureName1)=1是一个很重要的函数，将会在本安装程序内反复出现多次。这个函数用于判断用户是否选择了某feature。Help里对这个函数是这样描述的：FeatureIsItemSelected ( szFeatureSource, szFeature );

参数一：szFeatureSource，大意好像是feature的来源，具体不是很明白到底指什么，反正help自带的例子里写的MEDIA照抄没有错。

参数二：szFeatureName1，就是 feature的名字了

如果用户选择了这个feature，返回值就为1，往list里添加一个关于该feature的相关信息即可。

如此反复，判断所有的feature是否被选择，如被选择则添加一个相关信息即可。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image026_2.jpg)

这个就是显示了用户信息，安装路径和安装组件的信息。如果没有添加上述代码，这个界面默认是显示的，但是信息栏里是空白的。

顺便说一句，以前在制作这个安装程序的时候，因为这块显示是空白的，那时候对编程也是一窍不通的，情急之下，笔者把这个显示设置的框框设置了不可见。设置方法如下：

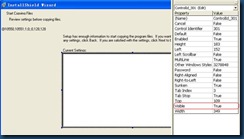
找到User Interface | Dialogs

[clip_image028](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image028_2.jpg)

在中间的导航树上找到SdStartCopy这个选项

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image030_2.jpg)

这里我们使用的是英文界面，因此点击选中English选项

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/ba11d68c5531_D534/clip_image032_2.jpg)

选中这个界面上的将会显示用户信息的框，把右边的Visible选项设置为False即可

小结：在Dlg\_SdStartCopy界面里，用户还可以设置左上角显示的标题和消息，szTitle = ""; szMsg = "";这两行代码如果赋值为空，则显示如图所示的默认信息，用户可以赋值成自己想要显示的信息。

4. 根据用户选择的组件，从外部文件夹拷贝相应的文件到安装目标路径的文件夹中

这个用途常见于配置文件和授权文件的应用，同一程序，授权给不同的用户，只需要不同的配置和授权文件。如果将配置和授权文件每次都打包在安装程序里，那么变更一个用户就需要重新打包一次，这是一个浪费时间和精力的行为。如果将授权和配置文件（当然内容是加密过的）放在外部文件夹中，每次安装的时候从这个文件夹中读取拷贝，那么会是一个比较通用型的安装程序。

另外，本程序的好几个feature用到了相同的库，如果直接在feature下加库文件也可以，但是每一个feature都加一次这个库文件夹，整个安装程序就会变得很庞大，因此比较理想的情况是选到了这个feature的时候从外部拷贝这些库文件。

这里我们先不包括文档这个feature的说明，文档feature另有详细说明。

1. 这个功能需要在OnFirstUIAfter()函数体中实现，选择After Move Data | OnFirstUIAfter选项，即在选择了移动哪些数据后这个操作生效。

[clip_image002](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/89267f3e5231_D66C/clip_image002_2.jpg)

2. 之前我们已经接触过了如何判断是否选择了某个Feature，这里也需要判断是否选择了某个Feature，并且根据这个Feature来拷贝对应的外部文件

首先定义一些需要的变量并且进行赋值，蓝色字体即为所定义变量和赋值语句

function OnFirstUIAfter()

//feature name

STRING szFeatureName1;

STRING szFeatureName2;

STRING szFeatureName3;

STRING szFeatureName4;

STRING szFeatureName5;

STRING szSrcFile1;

STRING szSrcFile2;

STRING szTarFolder1;

STRING szTarFolder2;

NUMBER nResult;

STRING szTitle, szMsg1, szMsg2, szOption1, szOption2;

NUMBER bOpt1, bOpt2;

begin

//feature 定义

szFeatureName1 ="Server";

szFeatureName2 ="Client";

szFeatureName3 ="Watch\_Portion";

szFeatureName4 ="Log\_Portion";

szFeatureName5 ="Report\_Portion";

//需要拷贝的源文件

szSrcFile1 = "Test\\lib\\\*.\*";

szSrcFile2 = "Test\\databaselib\\\*.\*";

//拷贝的目的地，目标文件夹

szTarFolder1 = "lib\\\*.\*";

szTarFolder2 = "databaselib\\\*.\*";

3. 对每一个feature进行判断，进行相应的文件拷贝

在OnFirstUIAfter（）的begin和end之间添加如下代码：

//copy the lib to the target ,copy the necessary file to the target

if (FeatureIsItemSelected(MEDIA, szFeatureName1)=1) then

CopyFile(SRCDISK^szSrcFile1, TARGETDIR^szTarFolder1);

CopyFile(SRCDISK^"Test\\configure\\title.gif", TARGETDIR^"Server\\ title.gif");

CopyFile(SRCDISK^"Test\\configure\\background.gif", TARGETDIR^" Server \\ background.gif");

CopyFile(SRCDISK^"Test\\configure\\configure.dat", TARGETDIR^" Server \\configure.dat ");

endif;

if (FeatureIsItemSelected(MEDIA, szFeatureName2)=1) then

CopyFile(SRCDISK^szSrcFile1, TARGETDIR^szTarFolder1);

CopyFile(SRCDISK^"Test\\configure\\configure.dat", TARGETDIR^"Client\\configure.dat ");

CopyFile(SRCDISK^"Test\\configure\\license.dat", TARGETDIR^" Client \\license.dat");

endif;

if (FeatureIsItemSelected(MEDIA, szFeatureName3)=1) then

CopyFile(SRCDISK^szSrcFile1, TARGETDIR^szTarFolder1);

CopyFile(SRCDISK^"Test\\configure\\configure", TARGETDIR^" Watch Portion \\configure");

CopyFile(SRCDISK^"Test\\configure\\license.dat", TARGETDIR^" Watch Portion \\license.dat");

endif;

if (FeatureIsItemSelected(MEDIA, szFeatureName4)=1) then

CopyFile(SRCDISK^szSrcFile1, TARGETDIR^szTarFolder1);

endif;

if (FeatureIsItemSelected(MEDIA, szFeatureName5)=1) then

CopyFile(SRCDISK^szSrcFile1, TARGETDIR^szTarFolder1);

endif;

4. 代码解释

if (FeatureIsItemSelected(MEDIA, szFeatureName1)=1) then

CopyFile(SRCDISK^szSrcFile1, TARGETDIR^szTarFolder1);

CopyFile(SRCDISK^"Test\\configure\\title.gif", TARGETDIR^"Server\\ title.gif");

CopyFile(SRCDISK^"Test\\configure\\background.gif", TARGETDIR^" Server \\ background.gif");

CopyFile(SRCDISK^"Test\\configure\\configure.dat", TARGETDIR^" Server \\configure.dat ");

endif;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FeatureIsItemSelected(MEDIA, szFeatureName1) 这个函数用于判断用户是否选择了某feature。Help里对这个函数是这样描述的：FeatureIsItemSelected ( szFeatureSource, szFeature );

参数一：szFeatureSource，大意好像是feature的来源，具体不是很明白到底指什么，反正help自带的例子里写的MEDIA照抄没有错。

参数二：szFeatureName1，就是 feature的名字了

如果返回值为1，则说明用户选择了这个feature

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

CopyFile(SRCDISK^szSrcFile1, TARGETDIR^szTarFolder1);

拷贝文件的函数。Help里是这样描述的：CopyFile ( szSrcFile, szTargetFile );

参数一：szSrcFile，源文件，可带路径，要带有扩展名的文件名。当这个文件带路径时，则从这个指定路径下拷贝指定的文件；如果是不带路径的，则直接从安装文件所在盘的盘符下寻找指定的文件来进行拷贝。如果要拷贝某个文件夹下的一系列文件，可以使用通配符。

参数二：目标文件，可带路径，要带有扩展名的文件名。当这个文件带路径时，则将文件拷贝到这个指定路径下；如果是不带路径的，则将文件拷贝到安装路径下。支持通配符。

小结：上面这段代码的意思是：如果用户选择了某个feature，则从安装程序所在的盘下面的一些文件夹下拷贝文件到目标路径下的一些对应文件夹下。这里记住拷贝文件一定要带上文件的全名，包括扩展名。

#### 5. 如果用户选择了文档feature，则把文档文件夹拷贝进来，并且对该文件夹进行遍历，为每一个文档创建一个在开始菜单下的快捷方式

1. 这个功能仍然在After Move Data | OnFirstUIAfter()的函数里实现

先定义一些变量并赋值，蓝色字体

function OnFirstUIAfter()

//feature name

STRING szFeatureName6;//feature名

STRING szSrcFile3; //需要拷贝的源文件

STRING szTarFolder3; //拷贝的目的地，带文件名

STRING szTarFolder4; //拷贝的目标文件夹，后面有一个函数要用到不带文件名的目标路径

STRING szDocFile, szDocFileName;// szDocFile，查找函数返回的查询得到文件名；szDocFileName，要查找的文件名

NUMBER nResult; //数字型变量，存放函数的返回结果

begin

//feature 定义

szFeatureName6 ="Document";

//需要拷贝的源文件

szSrcFile3 = "Docs\\\*.\*";

//拷贝的目的地，目标文件夹

szTarFolder3 = TARGETDIR^"Docs\\\*.\*";

szTarFolder4 = TARGETDIR^"Docs";//文档的存放路径，不带文件名

2. 仍然在begin和end之间的函数体内把下面的代码拷贝进去即可

if (FeatureIsItemSelected(MEDIA, szFeatureName6)=1) then //如果选择了此feature

if(CopyFile(SRCDISK^szSrcFile3, szTarFolder3)=0) then //那么把要拷贝的文件拷贝过去

nResult = FindAllFiles(TARGETDIR^"Docs", "\*.pdf", szDocFile, RESET); //对拷贝过去的文件进行查找，该函数会在第一个符合条件//的文件处停止

while (nResult = 0)

LongPathToQuote(szDocFile, TRUE );

ParsePath (szDocFileName, szDocFile, FILENAME\_ONLY);//对查找到的文件获取文件名

AddFolderIcon(FOLDER\_PROGRAMS^"Test\\Docs",szDocFileName, szDocFile, "", TARGETDIR^"Docs\\icons\\help.ico" , 0 ,"" , REPLACE ); //为该文件创建快捷方式，快捷方式的显示名就是刚才获取的文件名

nResult = FindAllFiles(TARGETDIR^"Docs", "\*.pdf", szDocFile, CONTINUE);//从上一个查找的位置继续向下查找，进行循环

endwhile;

endif;

endif;

3. 代码解释

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (FeatureIsItemSelected(MEDIA, szFeatureName6)=1) then

endif;

如果用户选择了文档feature，则进行一些相应操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(CopyFile(SRCDISK^szSrcFile3, szTarFolder3)=0) then

endif;

这里执行了两步操作：

第一步，从源盘的Docs文件夹下把所有文件都拷贝安装路径的Docs文件夹下，注意在定义变量的时候使用了通配符

第二步，如果拷贝成功，则返回值为0，那么进行下一步相应操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

nResult = FindAllFiles(TARGETDIR^"Docs", "\*.pdf", szDocFile, RESET);

查找目标文件夹下所有后缀名为pdf的文件，从文件夹的开始位置进行查找，查找成功则返回0。

这个函数在这里有一个巧妙的应用，因为这个函数会在查找到第一个符合条件的文件时就会停止继续向下查找，因此利用静态变量的传值不同，来实现对文件夹的全部查找。

Help里的解释如下：

FindAllFiles ( szDir, szFileName, svResult, nOp );

参数一：szDir，被查找的文件夹

参数二：szFileName，需要查找的文件的名字，支持通配符，例如\*.\*，\*.pdf，\*.doc

参数三：svResult，函数会在查找到第一个符合条件的文件时停止，返回这个符合条件的文件的文件名，带全路径和含扩展名的文件名

参数四：nOp， 静态变量。CONTINUE，从上一次查找的位置开始查找，这个特性我们呆会儿会用到；RESET，从文件夹的开始位置进行查找；CANCEL，释放被上一次的FindAllFiles查找的函数。在Windows NT系统下，需要在安装过程中使用带CANCEL的FindAllFiles来释放之前的查找，确保安装的正确性（因此我怀疑查找有bug，这个函数用来弥补这个bug…）。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LongPathToQuote(szDocFile, TRUE );

szDocFile为上一个函数查找到的第一个符合条件的文件名，带完整路径，这个LongPathToQuote函数加上这个文件名上的括号；否则下面一个函数无法解析不带括号的长文件名。

Help里的解释如下：

LongPathToQuote ( svPath, nParameter );

参数一：svPath，长文件名

参数二：nParameter，静态变量。 TRUE，为长文件名加上括号；FALSE，为长文件名脱去括号。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ParsePath (szDocFileName, szDocFile, FILENAME\_ONLY);

解析带路径的长文件名，返回文件本身的文件名

Help里的解释如下：

ParsePath ( svReturnString, szPath, nOperation );。

参数一：svReturnString为返回的解析过的文件名，

参数二：szPath，即被解析的长文件名

参数三：nOperation，静态变量，指定用何种方式来解析。这里使用FILENAME\_ONLY，也就说返回值为不带路径、不包含扩展名的文件名。这个文件名被下面一步用作显示的快捷方式的名称。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AddFolderIcon(FOLDER\_PROGRAMS^"Test\\Docs",szDocFileName, szDocFile, "", TARGETDIR^"Docs\\icons\\help.ico" , 0 ,"" , REPLACE );

创建一个快捷方式，使用指定的图标。

Help里的解释如下：

AddFolderIcon ( szProgramFolder, szItemName, szCommandLine, szWorkingDir, szIconPath, nIcon, szShortCutKey, nFlag );

参数一：szProgramFolder， 要创建的快捷方式所在的文件夹。这里FOLDER\_PROGRAMS指开始 | 所有程序，因此我们的快捷方式将会出现在开始 | 所有程序 | Test的Docs下；如果要添加到桌面上，可以设置为FOLDER\_DESKTOP；FOLDER\_STARTUP 指添加为启动项；FOLDER\_STARTMENU添加到开始菜单下。

参数二：szItemName，help里解释很晦涩，解释为要添加到文件夹下的图标的名称，即出现的图标旁边的那个字符串。其实就是我们常说的快捷方式的名称。这里填写被解析出来的那个不带路径也不带扩展名的文件名。

参数三：szCommandLine，全限定路径的文件名或文件夹名，可包含命令行参数。这里传入刚才查找到的文件名，包含路径、文件名和扩展名。读者可能注意到这个参数被做了一些预处理，这个处理也是折腾了几次才搞出来的，不同的操作系统默认路径也是有是否带引号的差别的，这里需要显式地指定一下，以免在不同操作系统上运行时引起不同的结果。

参数四：szWorkingDir，工作目录。Help里的解释如下：设置这个目录为你的应用程序文件所在的地方；要设置包含了应用程序的目录为工作目录，则可传一个空字符串给这个参数。这个参数一开始我并未理解其含义，不过传空字符串也没有出错；在后来经理提出新要求：允许用户自行选择是否在桌面上创建快捷方式时无意中明白这个参数的含义；请读者随便寻找一个自己计算机上的任意位置的快捷方式，右键点击选择“属性”，这个szWorkingDir就是属性面板上的“起始位置”，值为这个快捷方式所指的应用程序所在的文件夹的路径。至少在我试验的程序里，创建开始菜单的快捷方式和桌面快捷方式，这个参数要求的值还是略有不同的，开始菜单里创建，可以直接传空字符串；而桌面快捷方式，传控字符串总是会出错，查看属性面板里的“起始位置”值为空，因此手动地传了快捷方式所指向的应用程序的所在文件夹的路径，后面在“安装结束时允许用户选择创建桌面快捷方式”话题里有详细说明。

参数五：szIconPath，带全限定名的图标的路径，即包含路径、文件名和扩展名

参数六：nIcon。如果不是使用Windows图标的话，统统指定为0；Windows图标我没有研究过，Help里说可以指定为0，1，2，3…n我猜测是不是图标文件本身包含了多个图标，而我可以指定使用哪个图标？

参数七：szShortCutKey，热键，一般用不到。如果有需要可以设置为比如"Ctrl + Alt + 1"这种形式。

参数八：nFlag，静态变量，多个用途。这个程序里我们使用了REPLACE，即永远使用当前这个快捷方式的属性；RUN\_MAXIMIZED ，当从这个快捷方式登录程序时，程序界面最大化；RUN\_MINIMIZED，当从这个快捷方式登录程序时，程序界面最小化； NULL，无任何操作（不知道这个无任何操作适用于何种情况？）。

小结：这段代码的重点在于

1) 实现对文件夹下的文件的遍历。因为之前笔者的文档都打包在程序里，苦于文档的名称和数量常常变更，每做一次都要耗费人力物力，而且在光盘里仍然需要单独放置一个文档文件夹供用户在没有安装程序前的随时查看，重复打包安装使得安装内容容量巨大，以至于从刻录小光盘改成刻录大光盘，从VCD盘改成DVD盘。这段代码在用户选择了安装文档的条件下，对外部文件夹进行了拷贝，并且读取文件夹下所有的pdf文件（依次类推，只要设置了正确的过滤条件，可以读取文件夹下想要的文件）。难点就在于将文件夹下的文件一个个读取出来并且获取该文件的信息。

2) 对读取的文件创建快捷方式，这个难点在于8个参数的理解。我在互联网上搜索了一阵子，并且啃了一阵子help，但是可能自己外语水平不是很过关，以至于第四个参数没有完全理解到底是什么意思，所见的例子也很单调并且偷懒，能赋””的地方都给赋了””，无语~~~~

整个安装程序做下来这一段代码是最难的，FindAllFiles在Help里解释是当碰到第一个符合条件的文件就会停下来，因此如何读取全部文件，并且获取文件信息，代码的撰写也是费了很大的功夫，并且参考了别人的程序修改出来的。

#### 6. 在安装结束时，显示readme.txt文件

这是个很有用的设置，但是在InstallScript工程里不是默认自带的，因此也需要脚本编程实现。

这段代码的位置是在After Move Data | OnFirstUIAfter()的函数里实现的

1. 首先，在安装的时候把readme.txt文件从源盘拷贝到安装目录下。把这段代码拷贝到After Move Data | OnFirstUIAfter()的begin和end;之间即可。README.TXT文件放置在源盘的根目录下，并且在安装时拷贝到安装目录下。

CopyFile(SRCDISK^"README.TXT", TARGETDIR^"README.TXT");

这段代码意味着当安装执行的时候，这个文件总会被拷贝过去。

2. 创建一个Finish界面，并在界面上设置询问是否显示readme.txt文件的选项。

之前我们看到当我们第一次选取了After Move Data | OnFirstUIAfter()选项时，系统会为我们创建如下代码（当然不创建也不要紧，自己敲就是了）

这个就是结束界面。Installscript工程默认安装完毕后，界面直接消失，而不会出现一个带有Finish按钮的界面让用户点击了以后才结束整个安装过程。

这段代码就是创建了一个Finish界面了，我们要对这段代码进行改造，使之出现一个是否显示readme的选项。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/89267f3e5231_D66C/clip_image004_2.jpg)

把上图中从Disable(STATUSEX);起到SdFinishEx这行的代码，全部替换成如下代码：

Disable(STATUSEX);

ShowObjWizardPages(NEXT);

bOpt1 = TRUE;

bOpt2 = TRUE;

szMsg1 = SdLoadString(IFX\_SDFINISH\_MSG1);

szTitle="";

szMsg1="";

szMsg2="";

szOption1="Show Readme";

szOption2="";

SdFinishEx(szTitle, szMsg1, szMsg2, szOption1, szOption2, bOpt1, bOpt2);

if (bOpt1=TRUE) then

if(FindFile(TARGETDIR, "README.TXT", szDocFile)=0) then

LaunchApp ( WINDIR^"Notepad.exe" , TARGETDIR^"README.TXT" );

endif;

endif;

3. 代码解释

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Disable(STATUSEX);

使默认的安装设置对话框无效。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ShowObjWizardPages(NEXT);

顺序执行这个OnFirstUIAfter()的代码，如果参数为BACK，则逆序执行

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SdLoadString(IFX\_SDFINISH\_MSG1);

返回参数所关联的字符串值，这个参数应当是一个资源ID。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SdFinishEx(szTitle, szMsg1, szMsg2, szOption1, szOption2, bOpt1, bOpt2);

参数一：szTitle，即显示在界面上的左上角的标题，如果传空值，则显示默认值

参数二：szMsg1，安装结束的界面上允许最多有两个可选项，这个参数可以显示第一个选项的一些相关说明，如果赋空则不显示任何说明

参数三：szMsg2，解释同上

参数四：szOption1,选项名。这个是一个Checkbox，如果设置为空则不显示，如果赋值则显示一个Checkbox并且在这个Checkbox旁边显示这个所赋的简短值。

参数五：szOption2，解释同上。

参数六：第一个选项的状态，如果设置为TRUE，则第一个选项Checkbox默认为选中状态，FALSE则为未选中状态。

参数七：第二个选项的状态，解释同上。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (bOpt1=TRUE) then

判断是否选择了checkbox。如果用户选择了这个选项，则进行下一步操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(FindFile(TARGETDIR, "README.TXT", szDocFile)=0) then

为了保险起见，需要进一步判断一下这个readme.txt是否被拷贝进来了

Help里解释如下：

FindFile ( szPath, szFileName, svResult );

参数一：szPath，文件所在的路径，不包含文件名

参数二：szFileName，文件名，包含扩展名

参数三：szDocFile，返回的文件名

如果查找成功，则返回值为1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LaunchApp ( WINDIR^"Notepad.exe" , TARGETDIR^"README.TXT" );

打开readme文件

Help里没有对这个函数的专门的解释，但是有个例子，以至于我看了好几遍才看懂要表达的意思

参数一：应用程序，也就是你用什么工具来打开第二个参数指定的文件。我们这里用记事本打开，因此要引用一下Windows下自带的程序Notepad.exe，路径为WINDIR^"Notepad.exe" 。如果是一些不是Windows自带的程序，比如PDF，DOC，还需要从注册表里得到所安装的目标位置，从这个目标位置得到要用的工具。有兴趣的朋友可以试验一下。

参数二：要打开的文件，带路径，包含扩展名

小结：这个界面我曾经试图写在OnFirstUIBefore()里的结尾部分，用Dlg\_SdFinish来实现，但是总是发现虽然结束界面能出来，但是上一个界面不能消失掉的情况。因为这个资料也不好找，仓促之间试验出上述所说的办法，估计是等安装界面结束后补上一个界面来达到这个效果的；其实我本人是比较讨厌结束的时候有这么一个要看readme的选项的，一般自己装到这种软件，都是去掉钩选框，不看readme的；但是如果直接结束掉，不出这个结束界面又觉得提示不足，有时候不能确定安装程序有没有结束，所以私下里还是比较想去掉readme选项，而直接显示一个只有一个finish按钮的界面的。

#### 7. 在安装结束时，允许用户选择是否显示桌面快捷方式

有时候我们会看到别的安装程序在安装过程中允许用户选择是否要在桌面上显示快捷方式，一开始因为我们公司的分布式系统的组件太多了，不想显示在桌面上，而且觉得和在开始菜单中显示快捷方式的原理是一样的，因此也就轻轻带过；后来经理抱怨说没有桌面快捷方式，总是要去开始菜单找，觉得麻烦，而且客户是使用专用计算机运行我们的程序，也就是桌面上会很干净，希望我能够做这个功能出来。我试了一下，发现和在开始菜单中显示快捷方式还是有一点不同的，也是值得写出来的，至少可以让读者少走一些弯路。

1. 首先要显示一个允许用户选择是否显示桌面快捷方式的界面，这个界面上要有一个checkbox（钩选框），当钩选了以后，安装程序就要在安装时为用户显示桌面快捷方式。

这段代码的位置是在After Move Data | OnFirstUIAfter()的函数里实现的，也就是和“显示readme文件”的功能放在一起。

把从Disable(STATUSEX);起到SdFinishEx这行的代码，全部替换成如下代码：

Disable(STATUSEX);

ShowObjWizardPages(NEXT);

bOpt1 = TRUE;

bOpt2 = TRUE;

szMsg1 = SdLoadString(IFX\_SDFINISH\_MSG1);

szTitle="";

szMsg1="";

szMsg2="";

szOption1="Show Readme";

szOption2="Create Shortcut on Desktop?";

SdFinishEx(szTitle, szMsg1, szMsg2, szOption1, szOption2, bOpt1, bOpt2);

2. 代码解释

与上面的“显示readme文件”中的代码相比，只动了一个地方，即szOption2="Create Shortcut on Desktop?";

这个是一个Checkbox，如果值设置为空则不显示，如果赋值则显示一个Checkbox并且在这个Checkbox旁边显示这个所赋的简短值。

这里我们需要它显示出来，这样在界面上用户就会看到一个钩选框询问是否要显示桌面快捷方式。

3. 接下来我们要对用户所做的选择做一些判断，并且显示桌面快捷方式，在这段代码后面加上

if(bOpt2=TRUE) then

if (FeatureIsItemSelected(MEDIA, szFeatureName1)=1) then

szDocFile = TARGETDIR^"Server\\server.bat";

LongPathToQuote(szDocFile, TRUE );

AddFolderIcon(FOLDER\_DESKTOP, "Server" , szDocFile, TARGETDIR^"Server" , TARGETDIR^"Server\\icons\\appClient.ico" , 0 ,"" , REPLACE );

endif;

4. 代码解释

因为上面对这些函数的每个参数都有详细解释了，所以这里就不做一一解释了，只对要注意的地方做说明。

这里，一开始，笔者对第四个参数仍然传的是空字符串，但是创建的快捷方式总是不能运行，对比属性面板才发现，桌面快捷方式的“起始位置”的值居然是空的，看来Help解释的“当传空值的时候，默认为快捷方式所指的应用程序所在的目录”并未生效，只好老老实实地把运行目录的值手动地传进去。

读者可能注意到在AddFolderIcon函数里的第三个参数被做了一些预处理，这个处理也是折腾了几次才搞出来的，不同的操作系统默认路径也是有是否带引号的差别的，这里需要显式地指定一下，以免在不同操作系统上运行时引起不同的结果。

#### 8. 在安装结束后，启动指定的程序

在全部安装完毕后，启动指定的程序，向Windows安装一个服务。或者也可使用于安装结束后的程序的自启动。

1. 这部分很明显是要在安装全部结束后进行的，因此放在After Move Data | OnEnd里

[clip_image006](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/89267f3e5231_D66C/clip_image006_2.jpg)

2. 把OnEnd（）的代码替换如下

function OnEnd()

STRING szFeatureName;

STRING serviceTarget;

STRING szDocFile;

begin

/\*

//这个服务所需的文件只有在钩选了某feature时候才会被拷贝，并且也只有在用户钩选安装了此feature时候才会在安装结束时安装此服务，因此首要判断是否选择了此feature，然后寻找到该执行文件，并且进行安装

\*/

szFeatureName="Watch\_Portion";

serviceTarget=TARGETDIR^"watch.exe";

if (FeatureIsItemSelected(MEDIA, szFeatureName)=1) then

if(FindFile(TARGETDIR, " watch.exe ", szDocFile)=0) then

if (LaunchApp (serviceTarget, "") < 0) then

MessageBox ("Unable to launch "+serviceTarget+".", SEVERE);

endif;

endif;

endif;

end;

3. 代码解释

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (FeatureIsItemSelected(MEDIA, szFeatureName)=1) then

endif;

首先判断这个feature是否被用户选择安装。因为在这个应用程序里这个服务只与此feature相关，因此要做一下判断，如果用户没有安装这个feature，就不需要启动这个服务了。

当用户选择了这个feature时，返回值为0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(FindFile(TARGETDIR, " watch.exe ", szDocFile)=0) then

endif;

这个是判断一下文件是否被正确地拷贝过去了，这个文件应该位于安装目录下，名为watch.exe。当该文件存在时，返回值为0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if (LaunchApp (serviceTarget, "") < 0) then

endif;

启动该服务；如果启动失败，则返回小于0的值。

这里LaunchApp的用法和上面第6段的用法略有不同。这个函数的本意是启动第一个参数指定的运行程序来打开第二个参数指定的文件。这里第二个参数指定为空，因为没有要打开的文件；第一个参数指向我们需要启动的可执行程序即可。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MessageBox ("Unable to launch "+serviceTarget+".", SEVERE);

如果上一步中判断到程序未能正确启动，则弹出一个错误提示框体现用户。

小结：这段代码的用法非常简单，但是如果用在适当的安装程序里会非常重要；笔者的安装程序，在一开始的时候需要用户安装完毕后手动地去安装目录里找到这个服务并且启动，使人感觉非常不友好；现在在安装完毕后做到了静默启动，用户无需做任何事情。而且这个服务需要JDK的支持，配合上述第2段中判断是否安装了JDK这个应用，就不会出现安装了此服务但是无法运行的局面。

#### 9. 安装结束后，为JDK设置一个环境变量

之前提到了，要在安装本系统时判断是否安装了JDK，在最初笔者所做的安装盘中，还要让用户手动地去为JDK设置环境变量JAVA\_HOME，设置环境变量对于外行来说简直就是天方夜谭，在JAVA论坛新手区最常见就是求助设置环境变量的问题了，因此，这个功能最好还是由安装程序代劳为妙。

1. 这段代码在After Move Data | OnFirstUIAfter()里

//write the environment variable

szKey = "SOFTWARE\\JavaSoft\\Java Development Kit\\1.6.0\_04";

RegDBSetDefaultRoot(HKEY\_LOCAL\_MACHINE);

if (RegDBKeyExist(szKey)=1) then//如果该注册表值存在

if(RegDBGetKeyValueEx(szKey,"JavaHome",nvType,svValue,nvSize)=0) then//获取注册表值成功

szKey = "SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment";

if(RegDBSetKeyValueEx(szKey, "JAVA\_HOME", REGDB\_STRING, svValue, -1)<0) then

MessageBox ("Javahome create failed, please set it manually!", SEVERE);

endif;

endif;

endif;

2. 代码解释

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RegDBKeyExist(szKey)

判断JDK1.6.0\_04的注册表值是否存在；要判断JDK1.6.0\_04是否被安装，只有通过注册表来判断啦，同理可得，要是自己开发的一套系统中有多个安装程序，而且相互关联，就得朝注册表里写入值了。

如果返回值为1，则说明存在该键值；

如果返回值小于0，则说明该键值不存在。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

RegDBGetKeyValueEx(szKey,"JavaHome",nvType,svValue,nvSize)

因为设置JAVA\_HOME环境变量需要JDK的安装位置，所以要根据注册表来寻找到这个安装位置，而幸运的是，该键值下的JavaHome键名所对应的值就是JDK的安装位置。

Help里对该函数的解释如下：

RegDBGetKeyValueEx ( szKey, szName, nvType, svValue, nvSize );

参数一：szKey， 要查找的注册表的键，这里我们查找SOFTWARE\\JavaSoft\\Java Development Kit\\1.6.0\_04

参数二：szName，一些注册表键下面会有一些键名，如果你去看一下我们查找的键，会发现该键下存在多个键名，这里我们只要查找JavaHome键名对应的值，因此，指定szName为JavaHome

参数三：nvType，返回该键名对应的值的类型，比如字符型，数字型；当时笔者还犯了一个错误，以为这个参数是需要笔者指定类型的，因此写了一个REGDB\_STRING，结果编译出错，搞了半天发现这个参数是个返回值，汗一个。

参数四：svValue，返回该键名对应的值

参数五：nvSize，返回该键名对应的值的字节数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

szKey = "SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment";

RegDBSetKeyValueEx(szKey, "JAVA\_HOME", REGDB\_STRING, svValue, -1)

如果搜索注册表发现JDK已经安装了，就去读一下注册表的键值，并且设置我们所需要的环境变量，这两句话就是用来设置环境变量的。

环境变量也是利用注册表键值设置函数RegDBSetKeyValueEx来实现的，这个键是一个特殊的位置，一定是"SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment"，我们对该函数进行进行详细说明。

RegDBSetKeyValueEx ( szKey, szName, nType, szValue, nSize );

函数作用：设置注册表键值

参数一：szKey注册表里的键，这里，我们需要设置环境变量的值，因此这里固定传值为"SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment"

参数二：szName，键名，这里我们需要设置的是名为JAVA\_HOME的环境变量

参数三：nType，被设置的键的类型，这里是字符串型，并且不带%PATH%之类的符号，也不转行

参数四：szValue，就是键值了，这里我们已经从上面得到了JDK的安装路径，就把安装路径传进去

参数五：nSize，help里说明如果键类型为REGDB\_STRING, REGDB\_STRING\_EXPAND, 或者 REGDB\_NUMBER时，都可以设置该值为-1，installshield会自动为我们计算正确的长度，而当键类型为REGDB\_BINARY 和REGDB\_STRING\_MULTI时，就必须传该键值的实际大小进去。

小结：Installshield默认键值位置是在HKEY\_CLASSES\_ROOT下的，因此在这里，我们需要在进行搜索键值和设置键值的操作之前使用RegDBSetDefaultRoot(HKEY\_LOCAL\_MACHINE);这句话来设置一下默认的根键值为HKEY\_LOCAL\_MACHINE；另，在网上看了一个帖子，当时匆匆看了一下，说是设置的键值会在反安装时候卸载掉，我倒是没有在自己的安装程序里发现这个问题，不过可以研究一下；作者说当时为了解决这个问题，是在代码头加上DISABLE(LOGGING);代码尾加上ENABLE(LOGGING)来实现的，虽然我没有碰到这个问题，但是还是很感谢这位作者，因为当时他也说了，根本找不到资料，自己啃了天书般的HELP来解决，而自己一旦解决了问题，就分享出来，以便于大家少走弯路。

#### 10. 完美卸载

在第一部分的第9点我们提到过InstallScript工程里自带的Uninstall快捷方式的缺陷，这里我们将会创建一个可以实现全部卸载的卸载方式，这个卸载方式会以快捷方式出现在开始菜单下，利用安装程序本身的反安装功能来实现

3. 这段代码在After Move Data | OnFirstUIAfter()里，和其他创建快捷方式的代码放一起

function OnFirstUIAfter()

STRING szfilename,szFolder ,szmsg1,szmsg2;

NUMBER nresult;

begin

//创建删除快捷方式

szfilename = UNINSTALL\_STRING +" /UNINSTALL";

nresult = StrFind(szfilename,".exe");

if nresult >=0 then

StrSub(szmsg1,szfilename,0,nresult + 4);

StrSub(szmsg2,szfilename,nresult + 4,200);

LongPathToQuote(szmsg1, FALSE );

LongPathToQuote(szmsg2, FALSE );

szfilename = "\"" + szmsg1 + "\"" +szmsg2;

endif;

AddFolderIcon(FOLDER\_PROGRAMS^"Test","Uninstall",szfilename,WINDIR,"",0,"",REPLACE);

End;

4. 代码解释

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

szfilename = UNINSTALL\_STRING +" /UNINSTALL";

参数一：UNINSTALL\_STRING这个静态变量指向的就是我们的安装程序，也就是setup.exe，不过指向的位置不是我们的源盘里的setup.exe，而是C:\Program Files\InstallShield Installation Information\{0D9DF66A-44E5-4754-A522-2AD6C9D5CDBE}\setup.exe；Installshield创建的安装文件在安装时总会在这个文件夹里创建对应信息，一长串数字型序列码就是安装程序的Product ID。利用这个setup.exe就可以进行反安装

参数二：/UNINSTALL，告诉程序启动这个setup.exe时为非安装状态，即修复、重新安装和卸载状态。

因此，这个字符串的值应该是这种形式：

"C:\Program Files\InstallShield Installation Information\{0D9DF66A-44E5-4754-A522-2AD6C9D5CDBE}\setup.exe" -runfromtemp -l0x0409 /UNINSTALL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

nresult = StrFind(szfilename,".exe");

寻找到“.exe”这个字符串在szfilename这个字符串中的位置。

Help里对这个函数的描述如下：

StrFind (szString, szFindMe);

参数一：szString，被查找的源字符串

参数二：szFindMe，要查找的字符串

返回值为要查找的字符串在源字符串中的位置，如果返回值小于0，则说明源字符串中找不到要查找的字符串

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

StrSub(szmsg1,szfilename,0,nresult + 4);

StrSub(szmsg2,szfilename,nresult + 4,200);

如果要查找的字符串存在，那么源字符串就是正确的；这两句语句就对源字符串进行截断，得到想要的子串。

szmsg1应该为C:\Program Files\InstallShield Installation Information\{0D9DF66A-44E5-4754-A522-2AD6C9D5CDBE}\setup.exe

而szmsg2应该为 -runfromtemp -l0x0409 /UNINSTALL

Helpl里的解释如下：

StrSub ( svSubStr, szString, nStart, nLength );

参数一：svSubStr返回的结果字符串

参数二：szfilename源字符串

参数三：开始截断的位置。如果指定的位置大于整个被解析的字符串长度，则返回一个空字串。

参数四：结束截断的位置。如果指定的位置大于整个被解析的字符串长度，则默认为结束截断的位置是字符串的结尾处。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LongPathToQuote(szmsg1, FALSE );

LongPathToQuote(szmsg2, FALSE );

这两句的作用是对上面解析出的两个子串脱去括号。原本笔者参考的例子里没有这两句，在自己计算机上运行正常，但是换了一台计算机后，创建出的卸载快捷方式无效，查看快捷方式的指向发现和原来计算机的指向略有差别，查阅了一些资料得知Windows下的长文件名就有这个缺陷，每个操作系统解析出来的可能会有所不同，主要是引号的麻烦。在笔者自己的计算机上获取的长文件名是不带引号的，因此，解析正确；而测试的那台计算机上获取的文件名却是带引号的，这就造成了解析后拼凑的字符串的差别。这里就要显式地为解析出来的子串脱一下引号。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

szfilename = "\"" + szmsg1 + "\"" +szmsg2;

拼凑出正确的可执行文件的长文件名，带路径，包含扩展名

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

AddFolderIcon(FOLDER\_PROGRAMS^"Test","Uninstall",szfilename,WINDIR,"",0,"",REPLACE);

添加一个快捷方式到开始 | 所有程序 | Test下；照抄即可。

小结：可能读者会比较奇怪这一段代码的写法，因为中间那段if endif;代码看上去简直就是多此一举。在Installshield7之前，一直是这样写的：

szfilename = UNINSTALL\_STRING +" /UNINSTALL";

AddFolderIcon(FOLDER\_PROGRAMS^"Test","Uninstall",szfilename,WINDIR,"",0,"",REPLACE);

从Installshield8开始，长文件名一直有引号封闭不正确的问题，因此if endif;代码完全是为了解决这个问题而存在的，而上面提到的两个脱去引号的语句，是笔者在前人基础上修改加上的，因为发现解析出来的字串要是不脱一下括号还是有问题。

这个快捷方式运行的时候，出现界面和在安装完毕后再次运行安装程序出现的界面相同。选择Remove即可进行卸载。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/89267f3e5231_D66C/clip_image008_2.jpg)

这个卸载不会把程序运行时产生的文件卸载掉，比如日志文件、配置信息文件等；会把安装目录中所有从安装程序中安装的文件都卸载掉，包括安装时从外部拷贝的文件。利用Project Assistant创建的卸载快捷方式则无法卸载掉安装时从外部拷贝的文件。

#### 11. 完美卸载之卸载时触发命令（卸载Windows服务）

在做完这个安装程序后，以为可以结束了，没想到经理又提出了一个新的要求，因为之前的安装里（参阅第二部分的第8小节），在安装完毕后，启动了一个指定程序，这个指定程序干的事情就是向Windows写了一个服务进去（有兴趣的同学可以去看看Java Service相关资料，是一个把Java程序注册为Windows服务的一个工具或者说是组件更合适些）；所以，这里希望能够在卸载的时候能够把这个服务给卸载掉。

首先我们介绍一下两条Windows cmd命令：

1) SC stop XXX

这条命令用于停止某个名叫XXX的正在运行的Windows服务

2) SC delete XXX

这条命令用于删除某个名叫XXX的Windows服务

一开始我的思路是这样的，获取安装程序的卸载状态，然后调用这两条命令来删除服务；没想到这个“获取安装程序的卸载状态”让我浪费了整整一个下午的时间，只知道MAINTENANCE是程序的反安装状态，而这个反安装状态是有可能包括“重装”、“修复”和“卸载状态”的，当然我可以让反安装界面只能处于卸载状态，只要把前面创建卸载快捷方式中的szfilename = UNINSTALL\_STRING +" /UNINSTALL"; 这句话改成szfilename = UNINSTALL\_STRING +" /REMOVEONLY"; 就可以了；但是试验出来是不等我确认删除，这个服务就卸载掉了，原因是这个界面一出来就是MAINTENANCE状态，而程序捕获了这个状态后，是不管我是否按下了确认按钮就会去做这个操作了。

后来想在Onbegin里添加一个SdWelcomeMaint函数的判断，结果是判断倒是成功的，但是多了另一个重复界面。

看来这个思路可能是有问题的，然后满地google之，还是吞硬币的小猪的一篇文章给了启发，原文地址找不到了，只找到了这篇<http://school.ogdev.net/ArticleShow.asp?id=1699&categoryid=7>，这里面其实是谈反安装时候不执行OnMaintUIBefore函数的问题，我想既然这个函数是反安装时候“应该执行的”，那么就看看这个函数吧。

于是 打开Before Move Data | OnMainUIBefore

[clip_image010](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/89267f3e5231_D66C/clip_image010_2.jpg)

打开一看，大喜过望，这个函数里明明白白地显示了反安装时候的所有界面。

于是顺着向下看，找到Dlg\_SdFeatureTree。

**[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/89267f3e5231_D66C/clip_image012_2.jpg)**

这里红色圈出来的一行代码明确地告诉我们：如果为反安装状态，那么卸载所有组件！OK，代码只要添在这里就可以了。

[clip_image014](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/89267f3e5231_D66C/clip_image014_2.jpg)

这里就运用了一个函数LaunchAppAndWait来达到目的。其实一开始我还在想是不是要写批处理文件来执行呢，结果是不需要，直接写在这个函数里就可以了。

LaunchAppAndWait ( szProgram, szCmdLine, nOptions );

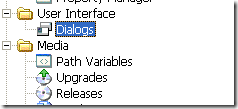
参数一：szProgram，要运行的程序。在Help里有这样一句解释：想在命令行里指定要运行的程序，那么可以对这个参数传空值

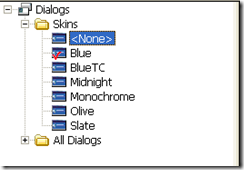
参数二： szCmdLine，命令行参数；很奇妙的参数，这里我们就可以写入我们想要的批处理语句了。

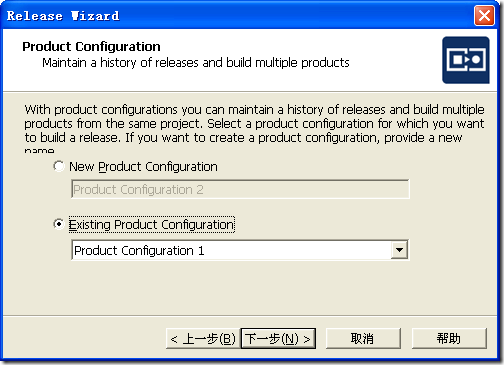
参数三：静态变量，操作类型，这里LAAW\_OPTION\_HIDDEN可以使批处理窗口隐藏掉，如果使用了LAAW\_OPTION\_WAIT，就会看到一个命令行窗口一闪而过，让人十分不爽。

于是，折腾了一下午的问题，就靠这短短的两分钟就解决了…

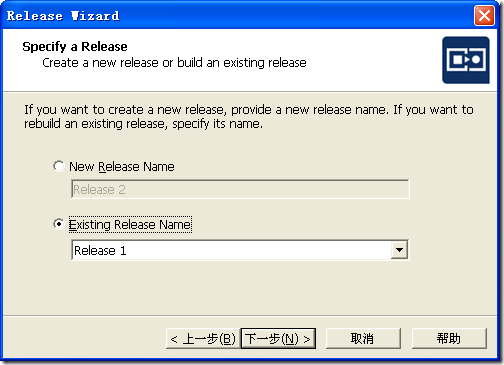
第三部分：其他   
1. 修改显示界面的风格   
Installshield 原始安装界面我始终觉得很丑，幸好Installscript 是可以不用写代码就可以改界面风格的。   
在 Installation Designer 的左边导航树上找到User Interface | Dialogs

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_2.png)   
在中间的导航树上找到 Dialogs | Skins

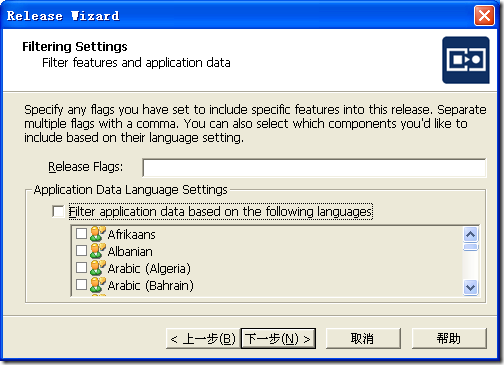
[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_4.png)   
Skins 选项下面显示了不同的界面风格，默认是None，选择一个喜好的风格即可，笔者一般使用Blue 或BlueTC，适用于一般商业软件的稳   
重风格；Midnight 比较酷，要是做电脑游戏的安装程序，我一定会选这个风格。   
2. 编译打包   
一切都准备就绪之后，就差一个编译打包成实际的安装程序的过程了。   
编译   
编译可以使用工具栏上的[WOTX[}}25C{DY637]~}(KXN](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/WOTX%5b%7d%7d25C%7bDY637%5d~%7d(KXN_2.jpg) ，检查一下有无定义错误，编译错误等。   
打包   
打包可以使用工具栏上的[image](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_6.png) 。   
1. 第一步，指定一个打包的配置版本，如果使用同一个安装程序源来打包成不同的配置，就可以选择新的配置版本。这里的配置指的   
是安装程序本身的配置，包括是否压缩文件、打包成网络安装程序或光盘安装程序、安装程序的客户信息、是否加密等等。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_8.png)

2. 同一配置下允许多个版本的存在；一般笔者习惯于当安装源文件升级的时候，就打包一个新版本出来以示区别，同时也可以保留老   
版本的安装程序备用不时之需。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_10.png)

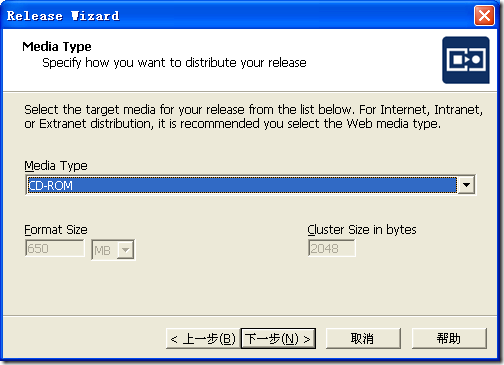
3. 过滤设置，笔者从来没用过。貌似是对feature 的一些设置参数，大概是针对某些版本，如果某些feature 不需要的话，就直接过滤掉   
了，安装时候无法选择安装了。不过参数具体怎么设置，包括下面的语言过滤，笔者尚未使用过。但是回想起以前安装的一些大型   
软件比如Oracle 之类的，会有一些选项是灰色无法选择安装的，大概就是类似这种功能。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_12.png)

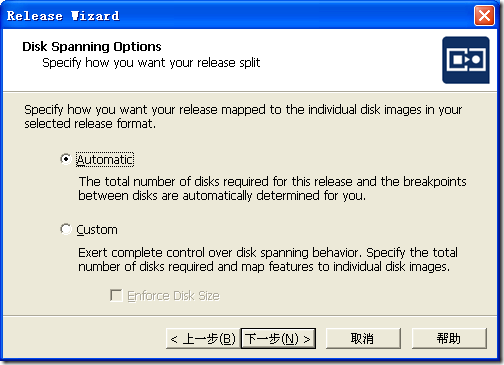
4. 安装程序的语言和被安装的应用程序的语言没有必然联系，因此这里还需要设置一下安装时所用的语言。选择列表里显示的语言取   
决于在做安装程序时选择的语言，刚才我们只选择了English，因此这里只显示English 了。如果选择了多项语言，那么在安装时出   
现的第一个界面会是让你选择安装时所用语言的界面，非常智能。这种功能针对需要发布到多个语种国家的软件是非常有用的。不   
过这样的话，在写脚本的时候，只要是显示在界面上的语言，除了系统可以默认显示的，都要多加一个语言判断，并且显示不同的   
内容了。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_14.png)

5. 选择介质类型，一般笔者都会把安装程序刻录到光盘上，因此选择 CD-ROM

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_16.png)

6. 光盘选项，第一个选项 Automatic 会自动为你检测所需光盘的规格数量，以及制作出光盘之间的断点。我一般都是选第一项，免去   
不少啰嗦事，第二个选项应该是高人才会选的吧。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_18.png)

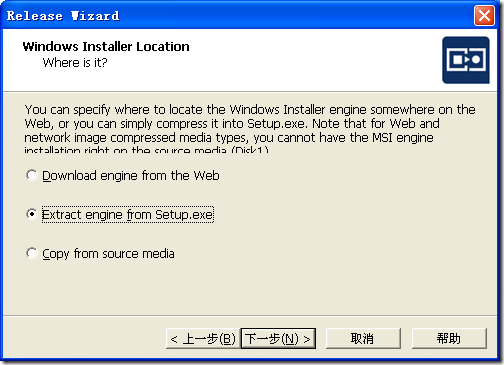
7. 这个选项指定了打包时的形态:   
Compress all files：所有的文件都压缩打包（这里不包括脚本里所写的安装时候从外部拷贝的文件，只包括在Project Assistant 里指定   
到各个feature 下的文件）   
Leave files uncompressed and separate from the installation package：所有的文件都不打包，以原始形态存在   
Custom：定制，允许你指定一部分压缩打包，一部分散放。   
笔者喜欢把所有的文件都打包，看上干净并且专业。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_20.png)

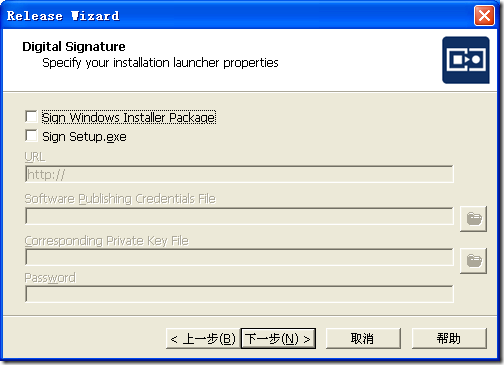
8. 对操作系统的要求，一般都默认即可

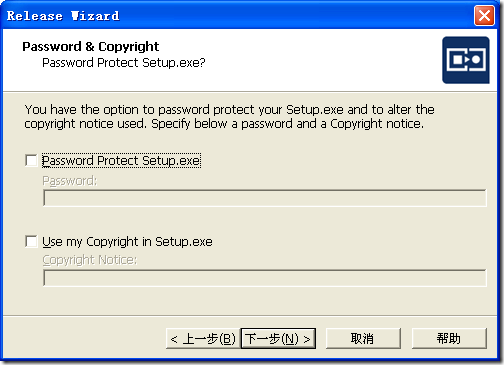
[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_22.png)

9. 这个是 Windows 安装引擎，对此没有研究过，凭着上面的解释选了第二个，会帮你打包进安装程序，这样就高枕无忧了。

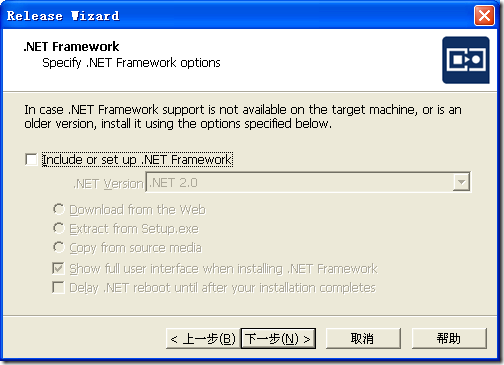
[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_26.png)

10. 签名，目前用不到，有兴趣的话可以导入数字签名文件

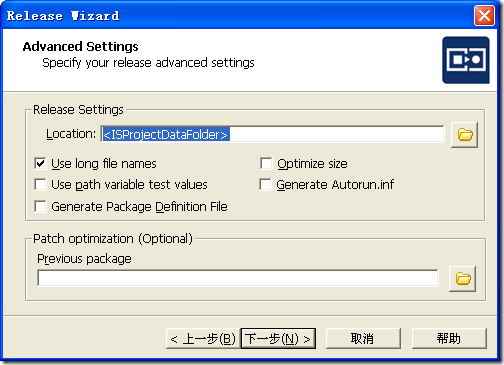
[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_28.png)    
11. 密码和版权声明。密码还是算了吧，一个商业软件是不该这样设的，客户会骂死的。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_30.png)

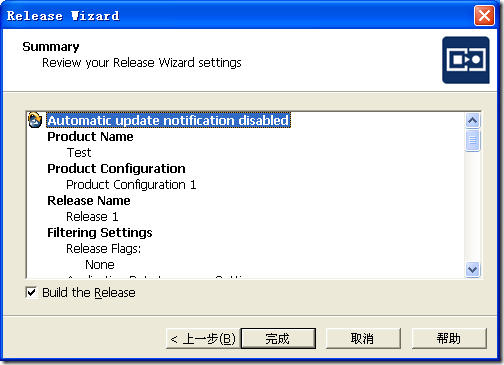
12. 是否要包括.NET Framework，Installshield 看来和微软很和谐啊。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_32.png)

13. 编译出来的文件放置的位置，以及一些相关设置，建议长文件名这个选项要选上，以免路径太深造成不必要的问题。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_34.png)

14. 最后一步，显示之前所配置的设置，点击“完成”即可编译出一个安装程序来。

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_36.png)

15. 编译好的安装程序文件应该在 工程路径\My Project Name\Product Configuration X\Release X\DiskImages\DISK1 下

[](http://images.cnblogs.com/cnblogs_com/Cindy_weiwei/WindowsLiveWriter/02cae52e7f6b_133B0/image_38.png)

点击 setup.exe，就可以安装了。   
这里顺便说一下，这个安装程序虽然制作好了，但是如果这样光秃秃地拿给客户，客户是要对产品的印象打折扣的。可以使用 Flash 或者其   
他的专业光盘制作软件来制作一个漂亮的应用界面来提供给客户，在插入光盘的时候自动弹出一个漂亮的使用界面，不但印象深刻，而且客   
户使用起来也方便。

### 第四部分：补遗

这篇补遗是《**一个完整的安装程序实例—艾泽拉斯之海洋女神出品**》的追加叙述，是在这个安装程序安装后发现应用程序运行时的一些小问题，通过安装程序中传递参数和设置环境变量来解决掉问题的，与安装程序本身无关。

这个程序做完之后，工程进入最后的调试阶段；里面发现了两个问题，其实和安装程序本身没有关系，但是都可以通过安装程序来解决，记录在此，以备以后碰到此类问题时可以查阅。

1. JAVA\_HOME的问题

之前在第二部分的第九小节里提到了安装完毕后，为JDK设置一个环境变量，事实上这个文档写到这里的时候有一点搞错了先后顺序，因为需要这个环境变量的是第二部分第八小节里安装完毕后需要启动的那个程序，所以后来调试时候发现了，就把第八节的程序内容和第九节的程序内容调换了一下顺序。

但是，很快发现了新问题，在注册表里添加环境变量和在桌面上“我的电脑”里直接添加环境变量是不一样的；注册表里的操作，都需要通过重启动计算机来使之生效，所以矛盾出现了：当这个程序启动的时候，环境变量还没有生效；而如果设置了让计算机重启动，就必须让客户手动启动这个程序，这是非常不友好的操作。

所以这里修改了一下方法，首先把需要启动的程序，也就是一个批处理文件，里面的

set JAVA\_HOME=%JAVA\_HOME%

这句话改成了

set JAVA\_HOME=%1

在批处理里面，需要从外部接收参数的时候，可以把参数写成%1，%2…%n。

然后，在第八小节的程序基础上修改，把

if (LaunchApp (serviceTarget, "") < 0) then

这句话修改成

if (LaunchApp (serviceTarget, javahome) < 0) then

即可。

2. 代码解释

这里javahome就是第九小节里if(RegDBSetKeyValueEx(szKey, "JAVA\_HOME", REGDB\_STRING, svValue, -1)<0) then这句话里的svValue，即JDK的安装路径。

LaunchApp (serviceTarget, javahome)

不能不佩服IS函数设计者，在这篇文档里，三个地方用到了LaunchApp这个函数，而且每个用法都不同。

参数一：这里写我们要打开的文件，带相对路径的

参数二：cmd\_line，这里，我们写入了JDK的路径，这个值将作为参数传递给我们要打开的批处理文件，批处理接收到JDK路径后，便可以正确启动了。

3. Path的问题

这个问题其实和JavaSerive以及操作系统相关；因为在笔者的计算机上一直没有发现这个问题。

在第二部分的第八小节中提到，我们会向Windows安装一个服务，但是笔者在工程用的计算机上始终不能启动这个服务，这次这个调试任务推给了经理，他经过多次试验，发现是环境变量中Path 的问题，只要在Path里添加上JRE的Bin文件路径，这个服务就可以正确启动了。因为没有深入了解JavaSerive的运行机制和Windows服务的运行机制，也没有深究为什么了，猜想可能是这个服务需要找这个路径，而有的操作系统只要指定了JAVA\_HOME就可以寻找JRE了，而有的却不行。

闲话休叙，我们需要写一段程序来实现这个功能。

要注意的地方有以下几点：

l Path往往已存在，并且里面有内容，因此不可以像设置JAVA\_HOME一样，而要考虑往已有内容中添加JRE路径，并且要考虑内容之间的分号问题

l 要考虑到卸载状态时，不能把Path卸载。说到这个问题，要提一下前面的第九小节，发现引文作者关于卸载时是否卸载自己添加的注册表键值的理解还是不正确的，Help里关于RegDBSetKeyValueEx有这样一句话However, the newly created key is not logged for uninstallation unless it is a subkey of a key already logged for uninstallation.也就是新创建的键值不会被日志记录了要反卸载掉，除非它有子键值被日志记录了要卸载

程序内容仍然添加在OnEnd()里，写在最后，如下：

szKey = "SOFTWARE\\JavaSoft\\Java Runtime Environment\\1.6.0\_04";//jre的键

RegDBSetDefaultRoot(HKEY\_LOCAL\_MACHINE);//设置一下根键

if (RegDBKeyExist(szKey)=1) then//如果这个键存在

if(RegDBGetKeyValueEx(szKey,"JavaHome",nvType,svValue,nvSize)=0) then //查找这个键的值

javaPath= svValue;

endif;

endif;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*以上为第一部分，以下为第二部分

//wirte the environment variable PATH

szKey = "SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment"; //环境变量在注册表中所在位置

javaPath=javaPath+"\\bin";//jre\bin的路径

if(RegDBGetKeyValueEx(szKey,"Path",nvType,svValue,nvSize)=0)then//如果Path存在

if(svValue!="") then

if(StrFind ( svValue, javaPath )<0) then//如果path还没有jre\bin的路径信息

svValue=svValue+";"+javaPath; //添加路径信息，此时要带上分号

endif;

else

svValue= javaPath;//如果键值为空，则直接添加即可，事实上键值为空的情况不会出现，这句话是无用的判断

endif;

else

svValue= javaPath; //如果没有这个键值，把值也直接添加进去，事实上这个情况也不会出现，因为path在操作系统安装完毕后就存在了，//除非你手动删除了，但是那样操作系统也会有问题了

endif;

if(!MAINTENANCE)then

Disable(LOGGING);

if(RegDBSetKeyValueEx(szKey, "Path", REGDB\_STRING\_EXPAND, svValue, -1)<0) then//添加或者重设键值

MessageBox ("Path create failed, please set it manually!", SEVERE);

endif;

Enable(LOGGING);

endif;

4. 代码解释

第一部分的目的在于找出JRE的安装路径，所有的函数之前都有解释，不再赘述

第二部分：

szKey = "SYSTEM\\CurrentControlSet\\Control\\Session Manager\\Environment";

环境变量，位于注册表的这个位置

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

javaPath=javaPath+"\\bin";

需要寻找的是JRE下的bin文件

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(RegDBGetKeyValueEx(szKey,"Path",nvType,svValue,nvSize)=0)then

如果Path存在并返回了值。

这里其实如果加上一个判断此键是否存在，代码会更加完善，不过操作系统装完之后这个键是一定存在的，这里笔者偷懒了

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(StrFind ( svValue, javaPath )<0) then

判断返回的键值里是否包含jre的bin文件夹的路径

StrFind（szString, strFineMe）

在源字串里查找是否包含指定的字串

参数一： szString，被查找的源字串

参数二：strFineMe，要查找的字串

如果包含要查找的字符串，则返回要查找的字符串在源字符串里的位置；如果查找不到则返回小于0的随机数字

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(!MAINTENANCE)then

判断一下安装状态，使之只有在非维护（修改，重新安装，卸载）状态使起作用

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Disable(LOGGING);

Enable(LOGGING);

这两句话作用分别是停止日志记录和使日志记录重新生效，这是从网上的一篇心得里抄录的，当日志停止记录时候，安装程序就无视了停止日志后的所有操作，这样保证键值不会在反安装时候被操作；其实本来上面一句if(!MAINTENANCE)then（非维护状态时操作）在原文是没有加上去，作者说只要不让日志记录到操作注册表，这个操作就不会被反安装掉，不过好像自己试了一下不行，加上一句，比较保险些，至少试验出来是没问题的。

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(RegDBSetKeyValueEx(szKey, "Path", REGDB\_STRING\_ EXPAND, svValue, -1)<0)

写注册表的键值。

Help里RegDBSetKeyValueEx的对应帮助里有这样一句话If the value data already exists, RegDBSetKeyValueExoverwrites it，也就是，如果键值存在，那么覆盖它。不过对注册表操作后，要把计算机重新启动才能生效；这一点和我们直接在“我的电脑”里操作环境变量是不一样的。

这里，第三个参数改成了**REGDB\_STRING\_ EXPAND**，因为在Windows 2003server下，path含有一个%SystemRoot%的相对路径，当时一开始使用了REGDB\_STRING，结果无意中发现所有的dos命令都用不了了，在dos下输入path一看，该替换成绝对路径的地方都没有替换掉，当时也是一头雾水，上CSDN一问，有人提醒说我应该看一下IS程序里键值设置时候的类型问题，跑回来一看果然设置有问题，REGDB\_STRING是不认识相对路径的，换成REGDB\_STRING\_ EXPAND就可以了。

5. 文件的只读性问题

用光盘装程序的时候发现一个问题，当从光盘上拷贝出文件的时候，文件会默认为只读格式，导致配置文件不能正常存储数据库信息，因此，在安装程序代码里拷贝完文件后，指定一下文件的属性

SetFileInfo ( szPathFile, nType, nAttribute, szValue );

此处用作SetFileInfo ( szPathFile, FILE\_ATTRIBUTE, FILE\_ATTR\_NORMAL, "" );