SYBEX Bonus Chapter

Group Policy, Profiles, and IntelliMirror for Windows® 2003, Windows® XP, and Windows® 2000

(Mark Minasi Windows® Administrator Library)

Jeremy Moskowitz

Web Chapter 1: ADM Template Syntax

ISBN: 0-7821-4298-2

Sybex Inc.
1151 Marina Village Parkway
Alameda, CA 94501
U.S.A.
Phone: 510-523-8233
www.sybex.com

1

ADM Template Syntax

Original material in this document was written by Derek Melber.

The biggest obstacle to creating your own custom ADM templates is overcoming the language barrier. Specifically, ADM templates have their own required syntax to prepare for these Registry changes. Whether you're a seasoned C++ coder or never wrote a "Hello, World" program in your life, the simple programming language shown here should not prove challenging.

The ADM templates have a simple structure, which can be classified into two primary categories:

- An ADM template contains the path to the Registry location for the custom change.
- An ADM template alters the Group Policy Editor interface so that the administrator can work with a GUI to make the appropriate changes.

This reference will teach you the structure and language of ADM templates through some simple examples.

If you're already an ADM Template pro, there's not much that's new for Windows XP and Windows 2003, but it is documented here. Specifically, look for the `SUPPORTED` keyword, especially in conjunction with the `#IF VERSION` statement.

Creating a New Custom ADM Template

In Chapter 5, you created your own custom ADM template and added it to the Group Policy Editor. Instead of creating your own new file, you could have added it to a Microsoft-supplied ADM template. Sure, it cuts down on the amount of ADM templates floating around your environment, but you risk Microsoft "adjusting" an ADM template in a service pack. For instance, Microsoft updated the `system.adm` template in Windows 2000 Service Pack 2 and Service Pack 4. So, a best practice is to create your own ADM templates from scratch.

Another major benefit to creating new ADM templates for your custom settings is that you can modularize the custom changes. This will be important as you begin to add these new templates to the Group Policy settings throughout your organization.

In the first example here, we'll keep with the theme we used in Chapter 2. That is, we'll develop new ways to modify our system sounds. We'll modify one specific example, the Exit sound for a Windows 2000 Professional, Windows XP, or Windows 2003 computer system. By default, this setting is not configured, but you can quickly and easily change all these systems' exit sounds with a simple Group Policy setting.

Remember that ADM files can contain two types of settings: policies or preferences. The setting we're going to manipulate (the Windows Exit sound) cannot be a *policy*; rather, it can only be a *preference*. Recall the main differences between a policy and a preference:

Policies Temporary Registry changes that are downloaded upon logon and wiped clean upon logoff. These are set to modify the Registry in specific Policies keys. Applications need to be coded to recognize the presence of the keys in order to take advantage of the magic of policies. These are displayed with a blue dot in the Group Policy interface. Windows 2000, Windows XP, and Windows 2003 ship with only policies.

Preferences Persistent Registry changes sent from upon high using the Group Policy Editor. These "tattoo" the Registry. They work like old-style NT System Policy. These are set to modify the Registry anywhere. Even if you load an ADM template that has preferences, the interface prevents you from immediately seeing them. This is a safety mechanism because of their "tattooing" nature. In order to see them, choose View ➢ Filtering ➢ "Only show policy settings that can be fully managed." Preferences are displayed with a red dot.

Again, the Registry entries used to modify Windows sounds are not coded to recognize the Policies keys, as described in Chapter 5. Therefore, if the underlying application (in this case, Explorer) is not smart enough to look in the Policies keys, you have no choice but to make the setting a preference.

Setting up a new custom ADM template is similar to creating the example in Chapter 5. Follow these steps:

1. Determine where the Registry setting is for the Exit sound for the computer. This setting is near the Start sound, but in a different Registry key:

   ```
   HKEY_CURRENT_USER\AppEvents\Schemes\Apps\.Default\SystemExit\.Current
   ```

2. Determine whether this Registry change is for the user or computer. You can clearly see from the Registry path that this is a user setting, because it is located in the HKEY_CURRENT_USER handle key.

You now need to build a new ADM template for the custom change. The syntax, which is detailed in the following sections, is rather straightforward. For now, don't be concerned about the syntax.

3. Open a new document in Notepad and add the following information. Figure 1.1 shows the resulting custom ADM code change.

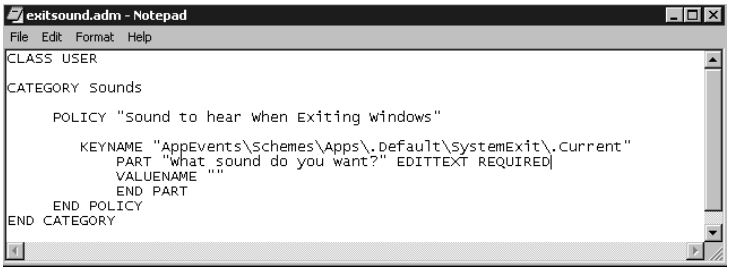
```
CLASS USER

CATEGORY Sounds

   POLICY "Sound to hear when Exiting Windows"

      KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
         PART "What sound do you want?" EDITTEXT REQUIRED
         VALUENAME ""
         END PART
   END POLICY
END CATEGORY
```

FIGURE 1.1 A custom ADM template with the Exit sound change as the only Registry change



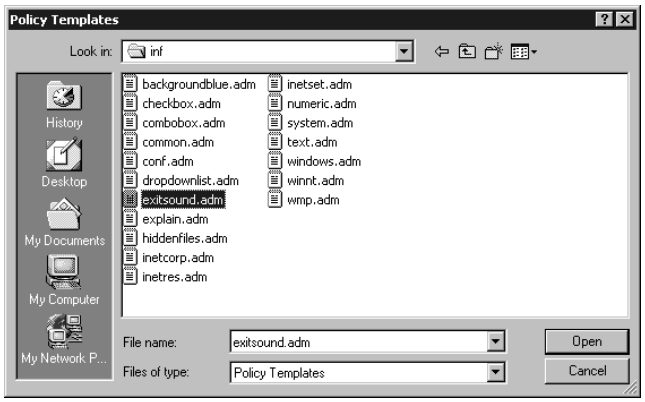
4. Save the new file. Here you have some liberty as to what name you want to give the template and where you want to store it. This ADM template was saved as `exitsound.adm` and stored in the *%systemroot%*\inf folder with the other ADM templates.

It is best to save all ADM templates in the *%systemroot%*\inf folder of your GPO management workstation (as described in Chapter 5).

After you create the new custom ADM template, you just need to add that template into the Group Policy Editor. Then, you can configure the Exit sound. For this step, you will need to import the new ADM template to the Group Policy Administrative Templates section of the GPO.

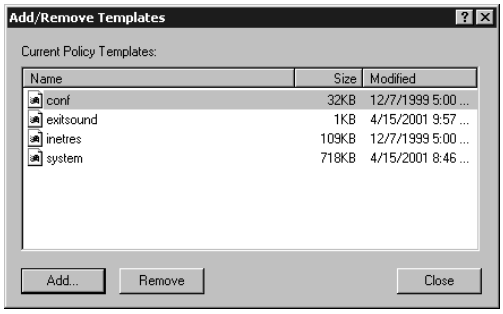
5. To import the ADM file, open the GPO you want to plunk this new ADM template in. Drill down to either Computer ➢ Administrative Settings and right-click the Administrative Templates and select Add/Remove Templates to open the Add/Remove Templates dialog box.
6. Click Add to add the template of your choice and open the Policy Templates dialog box, as shown in Figure 1.2. If you saved the templates to the INF folder, you can simply select the template. If not, browse to the folder and add the desired templates.

FIGURE 1.2 In the Policy Templates dialog box, you can select an ADM template to import.



7. After you select the template, close the Add/Remove Templates dialog box. The template is immediately available for you to configure. Figure 1.3 shows an imported template in the Add/Remove Templates dialog box.

FIGURE 1.3 The Add/Remove Templates dialog box with the new ADM template, `exitsound.adm`, imported



8. The Group Policy Editor displays the new setting for altering the Exit sound. Figure 1.4 shows this result, along with the previous ADM change for altering the Start sound.

If you do not see the red-dotted preference, be sure to uncheck the setting located at View ➢ Filtering ➢ "Only show policy settings that can be fully managed" as seen back in Chapter 5 of the book in Figure 5.5.

9. Now you need to configure the new ADM change. For this example, double-click the "Sound to hear When Exiting Windows Properties" preference. Here, we'll specify `chimes.wav`, as shown in Figure 1.5.

FIGURE 1.4 The Group Policy Editor showing the new custom Exit sound setting for our Windows client systems

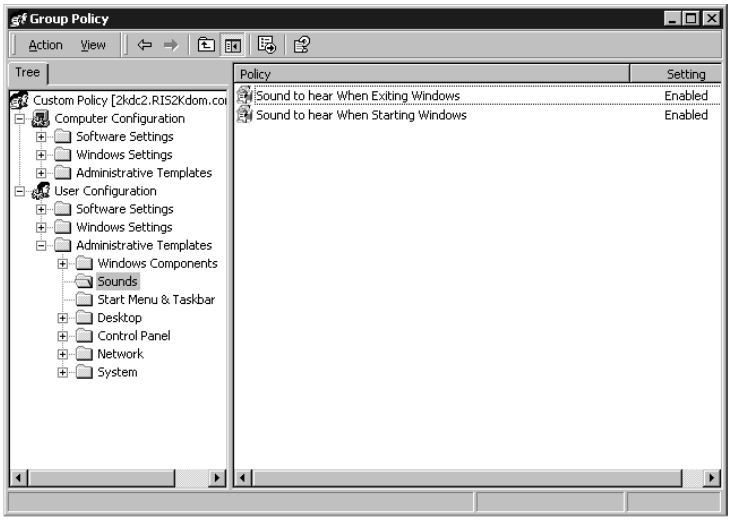
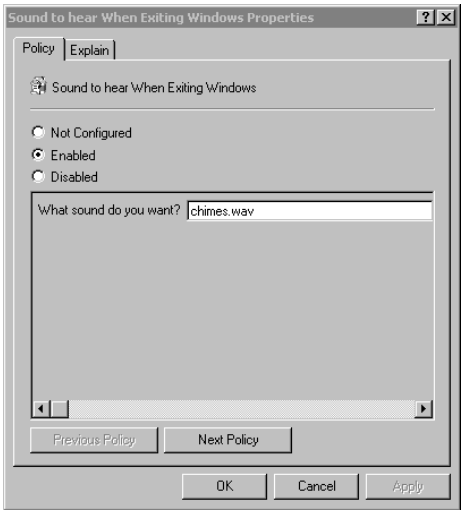


FIGURE 1.5 Double-clicking the properties of the policy setting allows you to set the custom Exit sound setting for our Windows client systems to `chimes.wav`.



As with the previous setting for the Start sound, you can change this to any WAV file on the client system. Or you can disable it, as shown in Figure 1.6, which turns off the Exit sound.

10. Ask the client to log on and download the setting that you have configured. Figure 1.7 shows the results when a Windows computer receives the preference. The Sounds And Multimedia Properties dialog box shows `chimes.wav` for the Exit sound.

FIGURE 1.6 Select Disable to disable this setting.

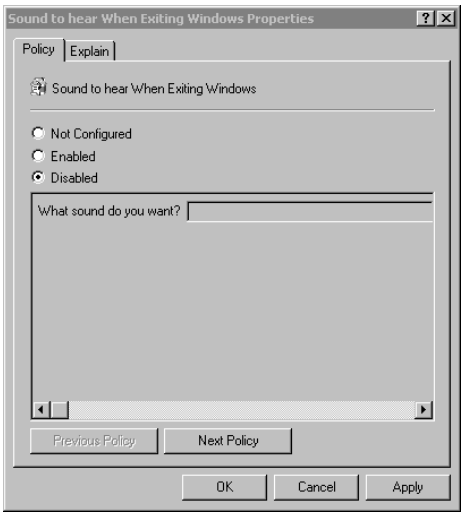
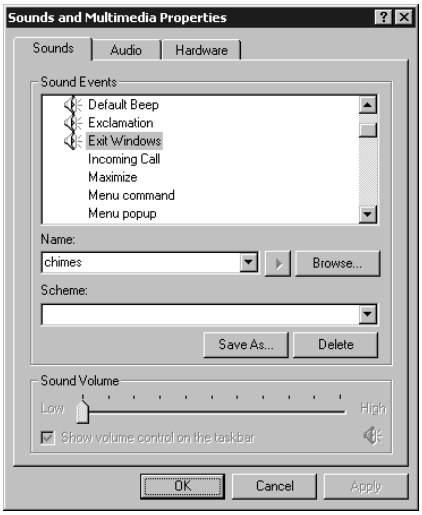


FIGURE 1.7 The Sounds And Multimedia Properties dialog shows our Windows computer is now set based upon the custom ADM change that alters the Exit sound to `chimes.wav`.



Variations on a Theme

Before we dive into all the syntax involved in a custom ADM setting, let's analyze the code used in Figure 1.1 (which shows the ADM template that was created for altering the Exit sound of a Windows computer system).

```
CLASS USER

   CATEGORY SOUNDS

      POLICY "Sound to hear when Exiting Windows"
         KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
            PART "What sound do you want?" EDITTEXT REQUIRED
            VALUENAME ""
            END PART
      END POLICY
   END CATEGORY
```

Recall that the Registry location for this change was the following:

```
HKEY_CURRENT_USER\AppEvents\Schemes\Apps\.Default\SystemExit\.Current
```

Before we get into our "formal training," let's inspect the elements in the ADM template shown in Figure 1.1.

CLASS USER Denotes that we will modify Registry settings in the HKEY_CURRENT_USER Registry handle.

CATEGORY The name we give the folder that appears in the Group Policy Editor interface. In this case, we're calling it Sounds.

POLICY The name of the policy we are assigning to this custom Registry setting.

KEYNAME The exact Registry path we will modify.

PART Designates that an input is required in the interface.

EDITTEXT The type of input allowed for this setting. We are using the REQUIRED modifier here to force an entry. (Other types of inputs and modifiers are explored later in this reference.)

VALUENAME The Registry value we will modify.

END PART Ends the PART section.

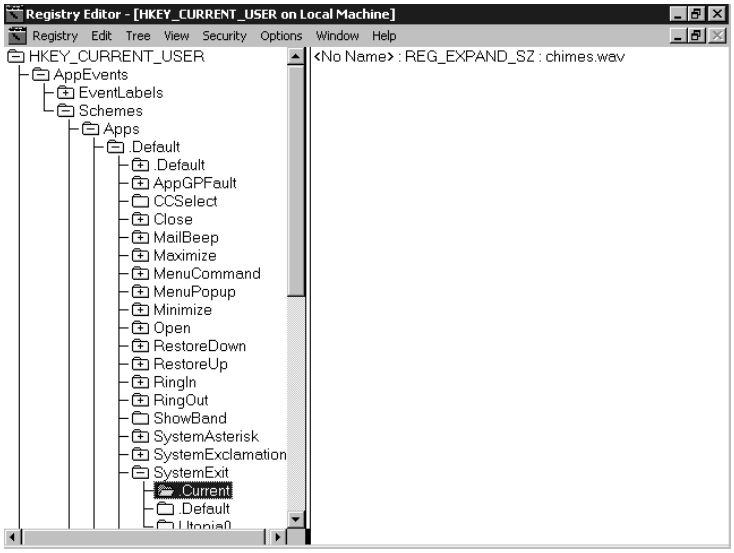
END POLICY Ends the POLICY section.

END CATEGORY Ends the CATEGORY section.

No END CLASS or END CLASS USER statements are required. Using these statements is considered a syntax error.

The VALUENAME keyword is followed by an empty set of double quotes. This is because this particular Registry entry is handled with a default entry value of <No Name>. In this case, we can represent this default entry in syntax form as " ". The actual Registry entry displays <No Name>, as shown in Figure 1.8.

FIGURE 1.8 The Registry entry for the SystemExit sound that was altered by the custom ADM template change, illustrating a value name of <No Name>



The CLASS, KEYNAME, and VALUENAME keywords specify the Registry definitions of the ADM template statements. Therefore, in the next section, we'll dive into these details. We'll then explore ways to accept input into the Group Policy Editor interface. For instance, we've already explored how an administrator can type a text entry into a field using the CATEGORY, POLICY, and PART keywords, but you can accept input in many other ways.

If you look at all the statements used in our example, you can see how they fit together to both generate the associated changes to the Registry and present the Group Policy Editor interface.

```
CLASS (Registry and Group Policy Editor)
   CATEGORY (Group Policy Editor interface)
      KEYNAME (Registry)
         POLICY (Group Policy Editor interface)
            PART (Group Policy Editor interface)
               VALUENAME (Registry)
```

The remainder of this section explores how to either update the Registry or modify how we take input via the Group Policy interface. With each new learned syntax, I'll modify the original code so you can see both what is being modified and how input is being taken.

The Syntax Used to Update the Registry

As previously stated, an ADM template understands three keywords that pertain to modifying the Registry:

- CLASS
- KEYNAME
- VALUENAME

The *CLASS* Keyword

When you begin creating your ADM template, you will need a CLASS entry section. This is the beginning of the ADM file. There are two options for type of CLASS:

CLASS MACHINE Indicates that all the Registry settings that follow modify the HKEY_LOCAL_MACHINE handle key in the Registry. Additionally, all these settings appear in the Computer Configuration node in the Group Policy Editor.

CLASS USER Indicates that all the Registry settings that follow modify the HKEY_CURRENT_USER handle key in the Registry. Additionally, all of these settings appear in the User Configuration node in the Group Policy Editor.

You can add multiple CLASS USER and CLASS MACHINE sections in a single ADM template. This could be beneficial if you want to organize your Registry settings within the same file, but not by user or computer. When the Group Policy processes the template, it merges all the common—USER or MACHINE—sections. In most cases, it is easier to track down the changes by keeping the CLASS USER and CLASS MACHINE sections together.

Later in this reference, we'll explore other syntax options that can be used when a CLASS statement is encountered. For that future reference, the statements that are compatible with the CLASS statement include CATEGORY and string variables, as discussed next.

The *KEYNAME* Keyword

Every policy that you create must have a KEYNAME entry. This entry is the exact Registry path for the change that you want to make on the end user's computer. The KEYNAME is typically located directly below the POLICY entry in the ADM template. If you want to make multiple changes to a single path in the Registry, add the KEYNAME entry after the CATEGORY entry. This allows you to add multiple VALUENAME keywords under this single entry of KEYNAME.

For example, when you want to modify the Legal Notice Caption and Legal Notice Text, you don't want to duplicate the KEYNAME, since they both use the same Registry path. If you need each Registry entry to be specific, add the KEYNAME entry below the POLICY or PART entries in the ADM file. In this scenario, you allow diverse settings and Registry paths, under the same CATEGORY listing. This allows you to update two totally different locations in the Registry, while still placing them under the same CATEGORY portion of the ADM template, which organizes them under the same folder in the Group Policy Editor interface.

The KEYNAME syntax includes the Registry path, without any leading backslash (\) or the handle key clarification. The handle key clarification is taken care of when a CLASS USER or CLASS MACHINE entry is encountered. The following is a sample KEYNAME entry:

```
KEYNAME AppEvents\Schemes\Apps\.Default\SystemExit\.Current
```

With all entries in the ADM template, you must include the text within double quotes if you have a space within the string. When the string does not contain any spaces, there is no need for double quotes (as in the previous example).

Do not add an END KEYNAME statement. The KEYNAME entry does not need or understand this statement.

The *VALUENAME* Keyword

VALUENAME is the actual Registry value being modified. When we modified the Exit sound for a Windows computer system, we had a value of "" in the ADM template. This was a unique situation—the actual value name in the Registry was <No Name>. This is the exception. In most cases, a VALUENAME other than <No Name> is usually present. In those cases, you'll simply enclose the VALUENAME value in quotes such as "legalnoticecaption", "legalnoticetext", or "dontdisplaylastusername".

As you investigate which Registry entries you want to modify, you will quickly see only two ways to perform the task:

- Some values are the ON or OFF type (typically a 1 or 0). These are possible with the VALUEOFF/VALUEON keywords.
- Some values require some form of text input. These are available with the PART keyword.

We'll explore the VALUEOFF/VALUEON keywords here; we'll save the syntax for the PART statement for the next section, because it affects the Group Policy Editor interface in such a distinct manner.

The *VALUEOFF/VALUEON* Keyword

The VALUEOFF/VALUEON keywords are similar to a light switch—on or off. (Also, 1 or 0; true or false; vanilla or chocolate—you get the idea.) The value in the Registry is either on or off. When you explore these values in the Registry, you'll notice that they usually consist of numbers, not text. Most Registry entries that fall into this category usually determine if the value is on or off. In most cases, an ON value is 1, and an OFF value is 0. Here is a (completely unrelated) example that illustrates the use of the VALUEON/VALUEOFF statements.

```
POLICY "Screen Saver"
  KEYNAME "Control Panel\Desktop"
```

```
    VALUENAME ScreenSaveActive
    VALUEON "1" VALUEOFF "0"
END POLICY
```

Some Registry values can handle an ADM entry that does not explicitly set VALUEOFF or VALUEON. In this case, the value is determined solely by the policy box options: Not Configured, Enabled, or Disabled. If the policy is set to Not Configured, the existing Registry value does not change. If the policy is set to Enabled, the value name is entered and has a data value of 1. If the policy is set to Disabled, the Registry value is deleted. Simply reenable to re-create the Registry value.

Do not add an END VALUENAME statement. The VALUENAME entry does not need or understand this statement.

Syntax Used to Update the Group Policy Editor Interface

In our Exit sound example, we used one statement, which modifies how we can enter values. Many statements are possible, and hence you can enter data in many ways. Here I'll go over the main statements that modify the Group Policy Editor interface and then discuss the remaining statements. The main statements include the following:

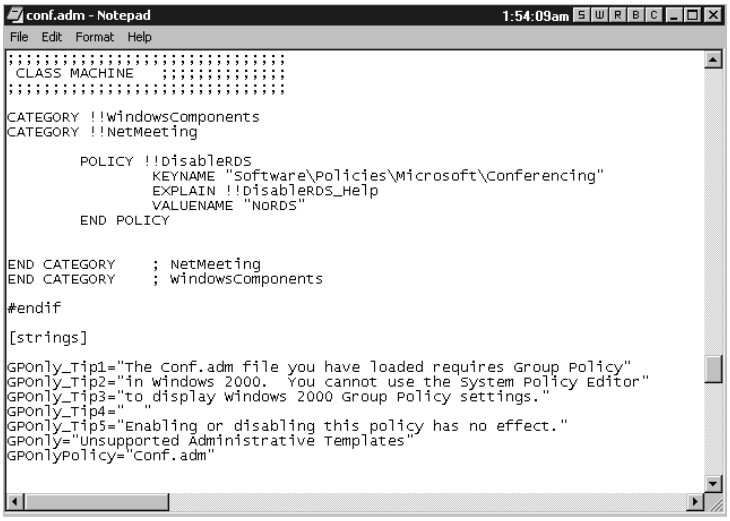
- STRING
- CATEGORY
- POLICY
- PART

The *STRING* Statement

The STRING statement is Microsoft's gift to you so that you can easily reuse portions of your ADM template code. Although our example did not contain any string variables, they are used extensively throughout the Microsoft default ADM templates, as shown in Figure 1.9.

The string variable is used in an ADM file to define text strings for the Group Policy Editor interface. By doing so, you commit to adding a [strings] section at the bottom of the ADM file. If you use the same text string in many locations in an ADM template, you can simply reference the same string and avoid the hassle of copying and pasting. Additionally, if you need to modify something, you can do so in one central location—within the [strings] section at the bottom—and all references that point to it are automatically updated.

FIGURE 1.9 UA snippet of the `conf.adm` template shows the string syntax within the main body of the ADM as well as the `[strings]` section.



With ADM templates, the string variable is extremely useful when creating `EXPLAIN` statements. You can use a shorter string variable in the main syntax and place the lengthy explanation below all the Registry changes, which allows for better viewing and organization within the actual ADM template file.

Later in this reference, we'll explore other syntax options that can be used when a `STRING` statement is encountered. Compatible statements include `CATEGORY`, `POLICY`, and `PART`.

The following code modifies our original structure so that we can reuse certain bits. In this case, we make references available to reuse Sounds, Exit, and SoundWav, which are replaced with "Sounds," "Exit," and "What sound do you want?" respectively.

```
CLASS USER
CATEGORY !!Sounds
  POLICY !!Exit
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
    PART !!SoundWav EDITTEXT REQUIRED
    VALUENAME ""      END PART
  END POLICY
END CATEGORY

[strings]
```

```
Sounds="Sounds"
Exit="Exit"
SoundWav="What sound do you want?"
```

The [strings] section contains variables. Each variable is not a statement—it contains the actual string value.

As you can see, you call a string variable by using two exclamation points (!!), followed by the string variable. The actual string value is in the [strings] section of the ADM template. This section is required since you used two exclamation points to specify the use of string variables and appears at the bottom of the ADM template.

If you decide to not use the string variable and opt to place the string value in with the code, you need to use a pair of double quotation marks around any string that has a space in it. We used this method in our original Exit sound example, as shown earlier in Figure 1.1.

The *CATEGORY* Statement

The result of using the CATEGORY statement in an ADM template can be seen in the Group Policy Editor interface as a folder. These folders hold the various policies that are contained below either the Computer Configuration or the User Configuration node.

The CATEGORY statement does not change the Registry and must be used in combination with other Registry-changing statements. The CATEGORY statement can be nested, though, resulting in a structure of folders that ultimately contain policies to be configured.

In our example, we labeled the CATEGORY statement Sounds.

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
      …
END CATEGORY
```

The CATEGORY statement must have a paired END CATEGORY keyword for every entry. This dictates which policies are contained within which folder. If you fail to supply an END CATEGORY keyword, the ADM import process generates an error message stating that additional code was expected.

Other statements can be used when a CATEGORY statement is encountered. Compatible statements include KEYNAME, CATEGORY and END CATEGORY, and POLICY.

The *POLICY* Statement

The POLICY statement helps denote the name of the new policy you are creating. The POLICY statement creates a policy in the pane on the right in the Group Policy Editor interface and typically follows the KEYNAME statement (which, as you'll recall, is the exact Registry path we're modifying).

Each KEYNAME statement can have more than one POLICY statement. The idea is that the policies listed under a single KEYNAME statement fall under the one Registry path indicated by the KEYNAME statement.

You can place the POLICY statement before the KEYNAME statement if you want the one policy listing to affect multiple Registry paths. This can be useful when you want to change many similar Registry values, even though they don't fall under the same Registry path. For example, you can change both the Start and Exit sounds under one policy. Since the Registry paths for both values are different, you can have a single POLICY statement, with two different KEYNAME entries, under the single POLICY statement.

In our example, we referred to the policy in the Group Policy Editor interface as Exit. This represented the Exit sound that we wanted to configure for all client machines affected by the GPO. A typical POLICY statement looks like this:

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
```

Other statements can be used when a POLICY statement is encountered. Compatible statements include KEYNAME, VALUENAME, VALUEON, VALUEOFF, POLICY, END POLICY, PART, ACTIONLISTON, ACTIONLISTOFF, and CLIENTEXT.

The *PART* Statement

The PART statement is a powerful portion of the ADM template. It specifies a label for the controls that are used to set a policy. This statement affects not only the Registry, but also the Group Policy Editor interface. In our example (see Figure 1.5 earlier in this reference), you can see that the PART statement places text below the policy settings of Not Configured, Enabled, and Disabled. If you look at the Exit sound ADM code, you will see that the PART statement directly follows the KEYNAME entry. END PART is required for all PART statements and indicates to the program that this portion of the ADM file is complete.

The PART statement can also be located below the POLICY statement if the POLICY statement follows the KEYNAME statement.

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
    PART "What sound do you want?" EDITTEXT REQUIRED
    VALUENAME ""
    END PART
  END POLICY
END CATEGORY
```

For a simple policy that only has a value range of 0 or 1 (off or on), you do not need to use a PART statement. You can simply use the VALUEON/VALUEOFF statements.

After you input this text, the Registry value at this location displays this text as the data under the value name <No Name>. As mentioned, the PART statement can encompass many options and controls. The example indicates two of those, which include EDITTEXT and REQUIRED. The EDITTEXT syntax allows for user input in order to enter the correct data for the value being changed. The REQUIRED syntax dictates that the policy cannot be implemented without a value being input for this PART.

Other statements can be used when a PART statement is encountered. Compatible statements include TEXT, EDITTEXT, NUMERIC, CHECKBOX, COMBOBOX, DROPDOWNLIST, LISTBOX, CLIENTEXT, PART, and END PART.

You just saw that the PART statement can have many uses. When you require the administrator's input to configure a Registry setting, the PART statement is a must. However, the PART statement cannot do everything that you want it to do alone. You can use many statements with the PART statement to allow for many types of interfaces, which allow for the necessary types of inputs. We will look at each type of PART that you can use to customize your Administrative Template interface.

The *TEXT* Type

This type displays a line of static text, which can also be referred to as a label. This will not update any Registry value; it will only display the label that you want . This is useful if you want to add a tip or an explanation about the Registry value that is near the PART statement.

Here is our Exit example used with the TEXT statement to clarify the details. (The results are shown in Figure 1.10.)

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
PART "This will modify the sound that you hear when Exiting your PC" TEXT
END PART
    PART "What sound do you want?" EDITTEXT REQUIRED
    VALUENAME ""
    END PART
  END POLICY
END CATEGORY
```

FIGURE 1.10 Use the "Text" type to specify some fixed text.

The *EDITTEXT* Type

This type offers the administrator an Edit Text field to input alphanumeric text. The text then updates a Registry value of type REG_SZ. The Exit sound example uses this statement.

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
```

```
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
    PART "What sound do you want?" EDITTEXT REQUIRED
    VALUENAME ""
    END PART
  END POLICY
END CATEGORY
```

In some cases, you will want more control over the EDITTEXT string. You can use the following options.

DEFAULT *value* The *value* option indicates the initial string to place in the Edit field. If you omit this option, the Edit field is empty by default.

MAXLEN *value* This *value* option specifies the maximum length of the input string.

REQUIRED This option requires that a string is entered in order to configure the policy properly. If you use this setting, an enabled policy cannot have a blank value.

OEMCONVERT This option sets the ES_OEMCONVERT style in the Edit field so that the string is converted properly from ASCII to OEM and then back to the Windows set. This option is extremely useful when edit controls require a filename.

Other statements can be used when an EDITTEXT keyword is encountered. Compatible statements include KEYNAME, VALUENAME, END, EXPANDABLETEXT, and CLIENTEXT.

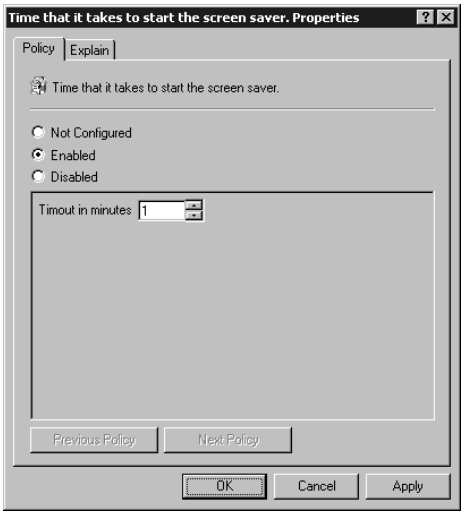
The *NUMERIC* Type

This type displays an Edit field that can accept a numeric value. This statement also has an optional spin control that allows a set range of numbers to be input that the final Registry value accepts. This can certainly help with Registry corruption and quality assurance with the values that are input. For example, the NUMERIC statement can configure the time it takes for the screen saver to start when the computer is idle.

```
CLASS USER
CATEGORY "Screen saver"
  KEYNAME "Control Panel\Desktop"
      POLICY "Time that it takes to start the screen saver"
      PART "Timeout in minutes" NUMERIC MIN 0 MAX 30 SPIN 1
        VALUENAME "ScreenSaverTimeOut"
      END PART
    END POLICY
END CATEGORY
```

The resulting Group Policy Editor change is shown in Figure 1.11.

FIGURE 1.11 The policy properties can include a spin control for the NUMERIC statement input.



The NUMERIC statement has other options that allow more control over the input.

DEFAULT *value* The initial value in the Edit field. If you omit the DEFAULT option, the Edit field is initially blank.

MAX *value* The upper limit for the numeric value. The default is 9999.

MIN *value* The lower limit for the numeric value. The default is 0.

REQUIRED Requires that a number is entered to configure the policy properly. If you use this setting, an enabled policy cannot have a blank value.

SPIN *value* Specifies the increment that the spin control uses for the numeric input. Increment by 5, 15, 50—whatever!

SPIN 0 Removes the spin control. The default value is SPIN 1, which activates the spin control.

TXTCONVERT Writes the value in the Registry as a REG_SZ string instead of a binary value.

Other statements can be used when a NUMERIC keyword is encountered. Compatible statements include KEYNAME, VALUENAME, END, MIN, MAX, SPIN, TXTCONVERT, REQUIRED, DEFAULT, and CLIENTEXT.

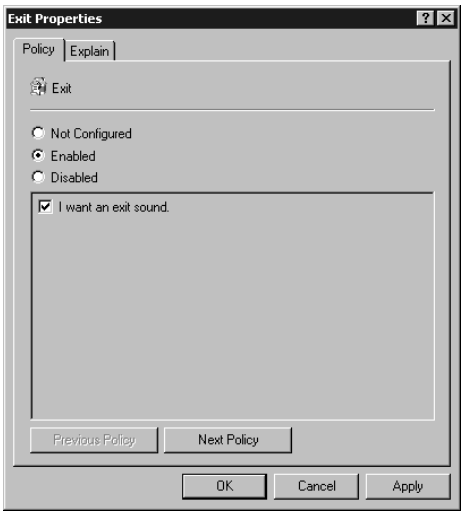
The *CHECKBOX* Type

The CHECKBOX type offers a check box in which the administrator can input the policy selection. The check box works like a light switch. When the box is selected, the default value is 1; when it is not selected, the value is 0. Additional options can control the input for the check box. One such option,

the VALUEON option, can make the Registry value almost anything. For example, the following ADM code slightly modifies the Exit sound example to use the CHECKBOX statement. Figure 1.12 shows the resulting Group Policy Editor interface.

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
    PART "I want an Exit sound." CHECKBOX     VALUENAME ""
    VALUEON "chimes.wav"
    VALUEOFF ""
    END PART
  END POLICY
END CATEGORY
```

FIGURE 1.12 The policy properties now shows the result of a CHECKBOX statement for the selection of chimes.wav for the Exit sound.



The CHECKBOX statement can take the following options.

ACTIONLISTOFF Specifies an optional action list to be used if the check box is cleared.

ACTIONLISTON Specifies an optional action list to be used if the check box is selected.

DEFCHECKED Specifies the initial setting of the check box selected.

VALUEOFF Changes the default check box setting to the setting specified.

VALUEON Changes the default check box setting to the setting specified.

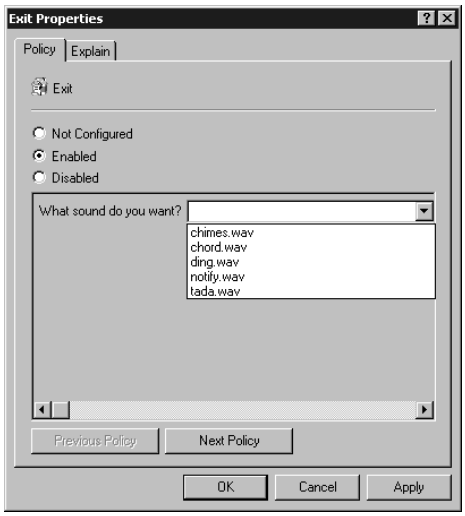
Other statements can be used when a CHECKBOX keyword is encountered. Compatible statements include KEYNAME, VALUENAME, VALUEON, VALUEOFF, END, ACTIONLISTON, ACTIONLISTOFF, DEFCHECKED, and CLIENTEXT.

The *COMBOBOX* Type

The function of the COMBOBOX statement is similar to that of the EDITTEXT statement, but the COMBOBOX statement can include additional options that offer a drop-down list of values. If you look at our example with the COMBOBOX statement included, the results are obvious. Figure 1.13 shows the resulting Group Policy Editor interface.

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
    PART "What sound do you want?" COMBOBOX
    VALUENAME ""
    SUGGESTIONS
      Chimes.wav Chord.wav ding.wav notify.wav tada.wav
    END SUGGESTIONS
    END PART
  END POLICY
END CATEGORY
```

FIGURE 1.13 The policy properties now shows the result of our COMBOBOX statement.



The COMBOBOX statement can take the following options.

DEFAULT *value* The *value* option indicates the initial string to place in the Edit field. If you omit this option, the Edit field is empty by default.

MAXLEN *value* This *value* option specifies the maximum length of the input string.

REQUIRED This option requires that a string is entered in order to configure the policy properly. If you use this setting, an enabled policy cannot have a blank value.

OEMCONVERT This option sets the ES_OEMCONVERT style in the Edit field so that the string is converted properly from ASCII to OEM and then back to the Windows set. This option is extremely useful when edit controls require a filename.

SUGGESTIONS Creates a list of possible settings for the Registry value. Remember to end the SUGGESTIONS option with the END SUGGESTIONS statement.

The *DROPDOWNLIST* Type

The DROPDOWNLIST type can be useful, especially if the Registry value that is being changed is not a simple ON or OFF switch or is a binary value that needs to be decrypted before the actual results are known. The DROPDOWNLIST statement helps because you want to see friendly text when making policy settings, not binary values.

This statement gives you a text string name that represents the actual value that is changed in the Registry. This is similar to the way that DNS names represent IP addresses. In the following revised Exit sound example, you can see how the DROPDOWNLIST type is used in this case. Figure 1.14 shows the policy that results from this setting.

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
    PART "What sound do you want?" DROPDOWNLIST
    VALUENAME ""
    ITEMLIST
      NAME "Xylophone" VALUE "Chimes.wav"
      NAME "Musical Chord" VALUE "Chord.wav"
      NAME "Door Bell" VALUE "ding.wav"
      NAME "Operatic Entrance Sound" VALUE "tada.wav"
    END ITEMLIST
    END PART
  END POLICY
END CATEGORY
```

FIGURE 1.14 The policy properties now shows the result of a DROPDOWNLIST statement.

The DROPDOWNLIST statement needs only the REQUIRED option. As you have seen with the other statements, this option requires that a selection be made before the policy can be established.

The *LISTBOX* Type

The LISTBOX type is designed to list multiple values that can be added or removed from the selection pool. The idea here is that some Registry entries can have multiple data values. One example might be a list of applications that the user can't run. This can be a powerful use of an ADM template and a Registry change. The following example shows the LISTBOX statement in action.

```
CLASS Machine
CATEGORY System
   POLICY Run
   KEYNAME Software\Microsoft\Windows\CurrentVersion\Run
      PART Runlistbox LISTBOX EXPLICITVALUE
      END PART
   END POLICY
END CATEGORY
```

For example, if you enter **notepad.exe** as the application, a value named notepad.exe whose data is notepad.exe is created at this location in the Registry. As you can see, the VALUENAME statement is not needed, since the text that you put in the policy becomes the value name in the Registry.

The following other options are available for the LISTBOX statement.

ADDITIVE When you include this option, the Registry values currently located at this target Registry are kept, and the new values are appended to them. The default action is to remove all values and replace them with the list that is specified by the newest policy entry.

EXPLICITVALUE When you specify this option, there will be two columns for the policy entry. One is the value name that appears in the Registry, and the other is the data. For example, if you enter **valuename = NOTEPAD**, the data is notepad.exe.

VALUEPREFIX *prefix* When you use this option instead of the EXPLICITVALUE option, you can specify the prefix for the value name. The prefix is then appended with an incremented integer for the complete value name. For example, if you select *application* as the prefix, the generated value names are application1, application2, and so on. If you want to display only the integer, you can specify a spacer (" ") that simply creates value names of 1, 2, and so on.

Other statements can be used when a LISTBOX keyword is encountered. Compatible statements include KEYNAME, END, VALUE-PREFIX, ADDITIVE, EXPLICITVALUE, EXPANDABILETEXT, and CLIENTEXT.

The *CLIENTEXT* Type

The CLIENTEXT type indicates which client-side extension (CSE) is needed for the Group Policy to process the correct settings on the client computer. These client-side extensions are DLL files that reside on the client machine and actually perform the Group Policy processing.

Client-side extensions (CSEs) are discussed in detail in Chapter 4. You can find all the client-side extensions in the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\GPExtensions Registry key.

The default behavior of the Group Policy is to process all settings configured under the Administrative Templates node. The CLIENTEXT statement alters the default behavior by specifying the extension to process these settings after the Registry extension has placed them in the Registry.

The following example shows how to use this statement using the Disk Quota ON and OFF switches.

```
CLASS MACHINE

CATEGORY !!DiskQuota
   KEYNAME "Software\Policies\MS\DiskQuota"
   POLICY !!DQ_Enable
      EXPLAIN !!DQ_Enable_Help
      VALUENAME "Enable"
```

```
        VALUEON  NUMERIC 1
        VALUEOFF NUMERIC 0
      CLIENTEXT {3610EDA5-77EF-11D2-8DC5-00C04FA31A66}
      PART !!DQ_EnableTip1    TEXT
    END PART
   END POLICY
END CATEGORY

[strings]
DiskQuota="Disk Quotas"
DQ_Enable="Enable disk quotas"
DQ_Enable_Help="Enables and disables disk quota management"
DQ_EnableTip1="Enable disk quotas for all NTFS volumes"
```

This code snippet not only illustrates how to use the CLIENTEXT type, but also shows how to tie together many of the statements discussed previously. Here are some things to note about the CLIENTEXT statement:

- It can be used only within the POLICY or PART scope of the ADM syntax.
- It should directly follow or precede the VALUENAME statement.
- The GUID (globally unique identifier) that is used with the statement is the GUID of the client-side extension (CSE).

Each GUID has a corresponding DLL. The GUID-to-DLL matching table for the CSEs is in Chapter 4.

Check out the Microsoft Platform SDK (www.microsoft.com/msdownload/platformsdk/sdkupdate/) for additional pointers on how to call the CSE DLLs.

Additional Statements and Syntax

You have seen almost all the statements that can be put in an ADM template. Some statements update the Registry, and other statements enhance the environment by changing the Group Policy Editor interface. But you can place even more elements in the ADM template. We have looked at some of these as options within other statements, and others are new. As you begin to build your ADM template, consider the overall goals. Consider the Registry changes as well as the interface changes you will use to implement the Registry changes.

After you get to this point, make your "statement of choice" and start to build the ADM template. As you are building, you might find some of these additional statements and syntaxes useful for clarity and detail:

- Comments
- EXPANDABLETEXT
- REQUIRED
- MAXLEN
- EXPLAIN
- #if Version
- SUPPORTED

Including Comments

As in any good coding environment, you want to leave a trail for the next person to follow when making changes or troubleshooting. You want to document the changes and the overall design of the ADM template within the template itself.

To add a comment to the ADM template, place a semicolon before the comment. You can place a comment on a new line or at the end of a valid line. As long as the comment follows the semicolon, the ADM file loads and runs successfully.

The following is our original code with some comments thrown in:

```
POLICY "Sound to hear when Exiting Windows"
   KEYNAME "AppEvents\Schemes\Apps\.Default\System\.Current"
      PART "What sound do you want?" EDITTEXT REQUIRED
      ; We need to make this a Required function
      VALUENAME ""
      END PART
END POLICY
```

The *EXPANDABLETEXT* Statement

The EXPANDABLETEXT statement is typically used in conjunction with the EDITTEXT statement and allows for system variables to be used in the Registry value (such as *%systemroot%*). This statement writes a value in the Registry that has a value type of REG_EXPAND_SZ. The following code continues with our Exit sound example. It changes a REG_SZ to a REG_EXPAND_SZ by simply adding the EXPANDABLETEXT statement to the PART statement.

```
CLASS MACHINE
CATEGORY TCPIP
  POLICY "Internet database file location."
  KEYNAME "System\CurrentControlSet\Services\Tcpip\Parameters"
```

```
    PART "Where are the files located (such as HOSTS, LMHOSTS, etc.)?" EDITTEXT
      REQUIRED EXPANDABLETEXT
    VALUENAME ""
    END PART
  END POLICY
END CATEGORY
```

Although the Group Policy Editor interface does not change, the value name in the Registry is now a REG_EXPAND_SZ value type.

The *REQUIRED* Statement

You use the REQUIRED statement when you want to enforce a policy to have a value that is valid. By default, some Registry entries choose a value of 0 or 1 if enabled, but not configured. To change this behavior, add the REQUIRED statement to many of the statements mentioned so that a blank entry is not allowed.

The *MAXLEN* Statement

The MAXLEN statement specifies the maximum length of text for a policy entry. This can be helpful when a Registry entry can recognize only a limited number of characters. If you don't use the MAXLEN statement, you could add a value that corrupts the Registry and causes serious damage.

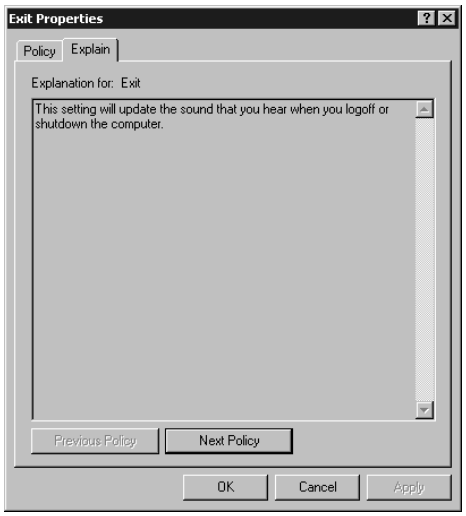
The *EXPLAIN* Statement

Using this statement lets you add help text to the Explain tab on the Group Policy Editor interface for the policy being modified. To access the Explain tab, you simply open the Properties dialog box for the policy in question. Our original Exit sound example did not contain an EXPLAIN statement, which is included in the following example.

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
  EXPLAIN "This setting will update the sound that you hear when you logff or
shutdown the computer."
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
    PART "What sound do you want?" EDITTEXT REQUIRED
    VALUENAME ""
    END PART
  END POLICY
END CATEGORY
```

If you apply this ADM template to a GPO, it returns the interface shown in Figure 1.15.

FIGURE 1.15 The policy's properties now has an Explain tab.



To create a new line within the explain text, you can use the \n syntax.) To create a space between two lines (creating a paragraph), simply use \n\n. The explain text itself is limited to a total of 4096 characters. It is recommended that you create the explain portion of the policy immediately after creating the custom entry to the ADM file. This helps others understand exactly what the purpose of the change.

The \n and \n\n syntaxes are valid only when used within the [strings] portion of the ADM template. If used within the quoted EXPLAIN keyword portion, \n and \n\n display as text, not as line returns.

Here are some tips to help you create good, clean, readable help text:

- Create a one- to two-line description of the policy. This should immediately indicate the intent of the policy.
- Describe the policy effects.
- Describe what the policy alters when it is enabled. This could include the Registry path and the value name.
- Describe what the policy alters when it is disabled.
- Describe what the policy alters when it is not configured. Be sure to be explicit, since different policies can have different effects when they are disabled.
- List the recommended policy settings, including most common scenarios.
- Provide special notes about the policy with regard to other policies and interactions.
- Include a list of other policies that rely on this setting to function.

The *#if Version* Statement

In your travels with ADM templates, you can conceivably use the same ADM template in a variety of places. You can write the ADM template to change the Windows Exit sound and then potentially use that file in Windows NT 4 System Policy, Windows 2000 Group Policy, or Windows 2003 Group Policy. In each version, the Group Policy Editor—where you enable or disable settings—has gotten smarter. To that end, this `#if Version` statement is useful when you want to write once and use anywhere.

To use this information, however, you need a chart of the version of the Group Policy Editors. So, here it is.

Group Policy Editor	Version Number
NT 4 and Windows 95 System Policy Editor	2
Windows 2000 Group Policy Editor	3
Windows 2003 Group Policy Editor	4

This statement is extremely useful for new ADM templates.

Much of the syntax in this reference is *version agnostic*. That is, you can use it in both old-style NT/9*x* System Policy as well as in Group Policy definitions. The `#if Version` statement is available only for Windows Group Policy ADM definitions.

The `#if Version` statement allows different versions of ADM syntax to be in the same file. To add an old template-setting syntax and new syntax to the same file , include a `#if Version` statement. Here is an example:

```
#if version >= 3
…
#endif
#if version <= 2
…
#endif
```

You'll see this used extensively if you open a Windows 2003 version of a supplied ADM template file, such as `System.adm`. Recall that in the Windows 2003 Group Policy Editor, you can see which versions of Windows a specific policy setting is directed toward: "At least Microsoft Windows XP," "At least Microsoft Windows 2000," or "At least Windows NetMeeting v3.0." That's a cool new feature that only version 4 of the Group Policy Editor can understand and display. Basically, version 4 of the Group Policy Editor is smarter than version 3 and can do more. Specifically, version 4 can use the `SUPPORTED` keyword (explored next).

If you take a look at the `Conf.adm` (the template used for NetMeeting), you can see how `#if Version` is used to leverage the `SUPPORTED` keyword, which only version 4 of the Group Policy Editor can understand.

```
; Chat and Whiteboard
     POLICY !!DisableChat

          #if version >= 4
               SUPPORTED !!SUPPORTED_NetMeeting3
          #endif

          KEYNAME "Software\Policies\Microsoft\Conferencing"
             EXPLAIN !!DisableChat_Help
             VALUENAME "NoChat"
     END POLICY
```

This is common in the new Windows ADM templates. It allows the new Group Policy to use the upper portion and the older Windows NT 4 System Policy to use the lower portion. Notice a couple of things in this example: the version number and the operators. The latest Windows NT 4 System Policy Editor is version 2. The first version of the Windows GPO snap-in is 3. Operators are used to compare the version numbers. You can use the following operators with the `#if Version` keyword:

`>` Greater than.

`<` Less than.

`==` (two equal signs) Equal to. For example, a==b is translated as "Is *a* equal to *b*?"

`!=` Not equal to.

`>=` Greater than or equal to.

`<=` Less than or equal to.

If you want to leverage the `SUPPORTED` keyword (explored in more detail next), you can simply say, "If the Group Policy Editor is smart enough to use it, leverage the `SUPPORTED` keyword." Here is that code:

```
CLASS USER
CATEGORY Sounds
  POLICY Exit
          #if version >= 4
               SUPPORTED !!SUPPORTED_Win2k
          #endif
  EXPLAIN "This setting will update the sound that you hear when you log off or
shut down the computer."
  KEYNAME "AppEvents\Schemes\Apps\.Default\SystemExit\.Current"
```

```
    PART "What sound do you want?" EDITTEXT REQUIRED
    VALUENAME ""
    END PART
  END POLICY
END CATEGORY
```

The *SUPPORTED* Keyword

The SUPPORTED keyword is new to Windows 2003 and Windows XP. As stated earlier, it works best in conjunction with the `#if Version` statement, since only the Windows 2003 and Windows XP Group Policy Editors understand and use it. As you can see in the previous code, you simply use the SUPPORTED keyword in conjunction with a string of text or a reference to other text via a text string variable. (See "The *STRING* Statement" section earlier in this reference.)

In our last code example, we said:

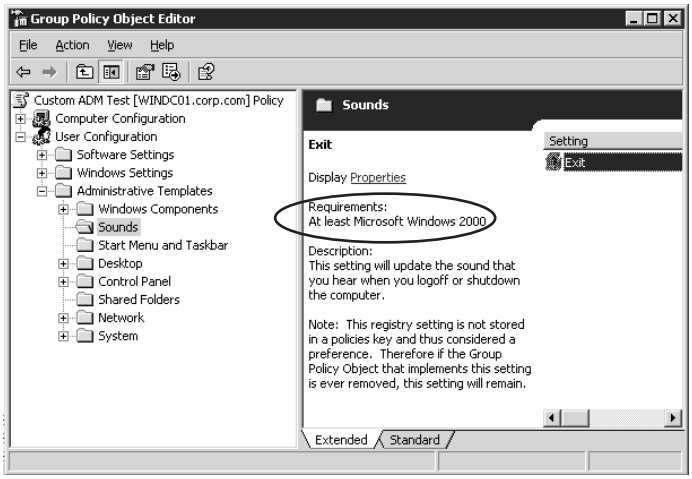
```
#if version >= 4
        SUPPORTED !!SUPPORTED_Win2k
    #endif
```

Therefore, all that's left is to ensure that in the `[strings]` section, you have a matching variable reference. For instance, your `[strings]` section might look like this:

```
[strings]
SUPPORTED_Win2k="At least Microsoft Windows 2000"
```

When you implement the SUPPORTED keyword in your code and then load the code into the Windows 2003 Group Policy Object editor, you can see the fruits of your labor, as shown in Figure 1.16.

FIGURE 1.16 You can see the SUPPORTED keyword in action here under the "Requirements" heading.



Practical Application of Custom ADM Templates

We've noodled through all the syntaxes possible for ADM templates. Along the way, we used those syntaxes and modified our ongoing example, the Exit sound, accordingly. Now, let's branch out a bit and see if we can harness this newfound power.

Example ADM Template: Custom Background

Have you ever wanted to change the background color of the desktop for the entire company's? Sure, you can use a built-in Group Policy to change the BMP of the background, but this is different. There's no way to change just the background color.

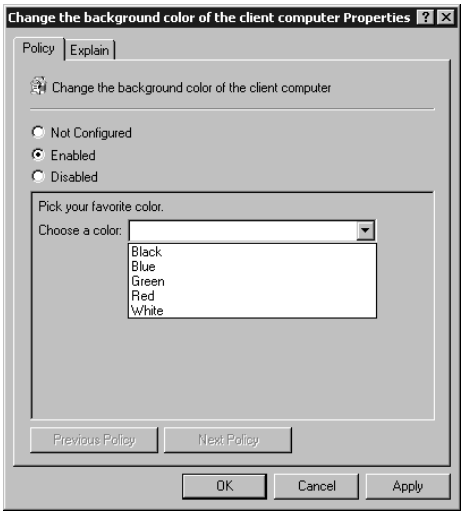
You can use the following ADM template to create a preference that changes the background to whatever color you desire:

```
CLASS USER
  CATEGORY "Background color options"
    POLICY "Change the background color of the client computer"
      EXPLAIN !!Explain_Colors
        KEYNAME "Control Panel\Colors"
           PART "Pick your favorite color." TEXT
          END PART

          PART "Choose a color:" DROPDOWNLIST REQUIRED
            VALUENAME Background
              ITEMLIST
                NAME "White" VALUE "255 255 255"
                NAME "Red" VALUE "255 0 0"
                NAME "Blue" VALUE "0 0 255"
                NAME "Green" VALUE "0 255 0"
                NAME "Black" VALUE "0 0 0"
              END ITEMLIST
          END PART
  END POLICY
END CATEGORY
```

Figure 1.17 illustrates the resulting Group Policy Editor interface for this ADM template.

FIGURE 1.17 You now have a custom policy to change the background color of the client computer.



Example ADM Template: Custom Windows Explorer Options

We all know that Microsoft was trying to do us a favor by preventing users from seeing the file extensions in Windows Explorer. This is also true with hidden files and the protected operating-system (or "super-hidden") files. However, as an administrator or a power user, it is extremely annoying to have to open Windows Explorer, choose Tools ➢ Folder Options to open the Folder Options dialog box, and then click the View tab to enable these three settings for every user who is in the know (including yourself).

Well, you will no longer have to do this! You can use the following ADM template to change these settings for whomever you feel should have them turned on by default:

```
CLASS USER
  CATEGORY "Windows Explorer options"
    POLICY "Enable the essential File settings for administrators"
      EXPLAIN "This policy will modify three different portions of the Windows
Explorer interface.The first option will either show or hide hidden files and
folders.The next option will show the Superhidden files, which are the protected
OS hidden files.The final option will allow the file extension to be seen or
not."
        KEYNAME
"software\Microsoft\windows\currentversion\explorer\advanced"
```

```
        PART "Do you want to see hidden files?" TEXT
        END PART

        PART "Hidden Files:" DROPDOWNLIST
          VALUENAME Hidden
            ITEMLIST
              NAME "Yes" VALUE NUMERIC 1
              NAME "No" VALUE NUMERIC 2
            END ITEMLIST
        END PART

        PART "Do you want to see Super Hidden files?" TEXT
        END PART

        PART "Super Hidden:" DROPDOWNLIST
          VALUENAME Showsupperhidden
            ITEMLIST
              NAME "Yes" VALUE NUMERIC 1
              NAME "No" VALUE NUMERIC 0
            END ITEMLIST
        END PART

        PART "Do you want to see the file extensions?" TEXT
        END PART

        PART "File extensions:" DROPDOWNLIST
          VALUENAME HideFileExt
            ITEMLIST
              NAME "Yes" VALUE NUMERIC 0
              NAME "No" VALUE NUMERIC 1
            END ITEMLIST
        END PART
  END POLICY
END CATEGORY
```

Figure 1.18 shows what the resulting Group Policy Editor interface will look like with the example ADM template imported.

FIGURE 1.18 You now have a policy to change the behavior of the file extensions and hidden files in the Windows Explorer interface.

Final Thoughts

The syntax of ADM templates is basic but powerful. Be sure you know which Registry settings you're modifying to ensure that you're not doing any harm to the target system. And be sure to try these in a test lab before rolling out changes to your clients.

At `www.GPOanswers.com`, you'll find a slew of additional, quite useful ADM templates to download. Likewise, if you have one you want to share with us, please do so!

Microsoft has its own white paper on the subject of ADM template files. Be sure to check it out at `www.microsoft.com/WINDOWS2000/techinfo/howitworks/management/rbppaper.asp`.