



Operating System

Implementing Registry-Based Group Policy for Applications

White Paper

Abstract

This white paper focuses on implementing registry-based Group Policy for applications that you are developing. This document begins with some details on what registry-based policy is, and when to use it. From there, the steps to develop registry based policy are described. The appendix to this white paper contains a full language reference to the .adm language used to deploy registry based policies.

© 2000 Microsoft Corporation. All rights reserved.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This white paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT.

Microsoft, Active Directory, IntelliMirror, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Microsoft Corporation • One Microsoft Way • Redmond, WA 98052-6399 • USA

08/00

Contents

Introduction.....	1
Overview of Registry-Based Policy	2
Policy versus Preference	2
Defining Policies and Preferences	2
How to Use Both Policy Settings and Preferences	3
When to use Registry-Based Group Policy	4
Designing and Developing Registry-Based Policy	6
Design Considerations	6
Best Practices for When to Create Group Policy	6
Best Practices for User Interface Design	8
Creating Explain Text for Group Policy	8
.adm Files	8
Best Practices for Developing Registry Based Policy Settings	9
When to Create a Policy	9
When Not to Create a Policy	9
Writer's Role in Developing Registry-Based Group Policy	10
Creating Names for Group Policy	10
Creating the Explain Text	10
Developers Role in Developing Registry-Based Policy	12
Testing Registry-Based Policy	17
Building Your .adm File.	18
Loading your .adm File Into the Group Policy Snap-In	18
Appendix A: .adm Language Reference	20
Sample .adm File	20
Components of the .adm Language:	21
Comments	21
Strings	21
CLASS	22
CATEGORY	23
POLICY	24

Appendix B: Additional Keywords41

Appendix C: Other .adm Topics.....45

Using Simple Policies, and Policies with the VALUEOFF and VALUEON Statements 45

EXPLAIN 46

Line Breaks 46

#if Version (for Version Comparison) 46

Changing the .adm Files Being Used for a GPO 47

Duplicate Category Sections 47

Summary49

For More Information 49

Figure 1. Example of .adm code and results in Group Policy. 20

Figure 2. Example of CATEGORY..... 24

Figure 3. Example of POLICY..... 25

Figure 4. Example of Group Policy. 26

Figure 5. Example of PARTS. 27

Figure 6. Example of different Part types..... 29

Figure 7. Example of EDITTEXT and TEXT PART..... 32

Figure 8. Example of CheckBox..... 34

Figure 9. Example of NUMERIC part. 37

Figure 10. Example of DROPDOWNLIST..... 39

List of Tables

Table 1. Components of Group Policy.	1
Table 2. Results of Group Policy Settings and Preferences.	3
Table 3. Policy Part Types.	27
Table 4. Options for EDITTEXT part types.	31
Table 5. Options for NUMERIC Part Types.	35
Table 6. Option for DROPDOWNLIST Part Type.	38
Table 7. Options for LISTBOX Part Type.	40
Table 8. Variants for ACTIONLIST used with Policy and Checkbox.	41
Table 9. Example 1 Policy Defaults.	45
Table 10. Example 2 Policy Defaults.	46
Table 11. Valid Operators for the Version statement number.	47

Introduction

Group Policy is the foundation of the IntelliMirror™ management technologies in the Windows 2000® Server operating system. To make use of all of its features, Group Policy requires Active Directory and Windows 2000 clients. In this environment, it allows an administrator to define and control the state of computers and users in an organization. Group Policy may be set on the following containers of the Active Directory: Sites, Domains, and Organizational Units (OUs). Additionally, the effect of Group Policy may be filtered using memberships in security groups. Once set, the system (Group Policy) maintains that state without further intervention.

The following table lists the components of Group Policy.

Table 1. Components of Group Policy.

Component	Description
Administrative Templates	Registry based policy, known as System Policy in Windows NT® Server 4.0.
Security Settings	Security settings for domains, computers and users.
Software Installation	Assign or publish applications.
Internet Explorer Maintenance	Administer Internet Explorer after deployment.
Scripts	User logon/logoff and computer startup/shutdown.
Folder Redirection	The ability to re-direct folders and files to the network.

This document focuses on how you can implement registry-based policy for your application. The following topics are presented:

- Overview of registry-based policy.
- When to use registry-based policy.
- How to design, develop, and test registry-based policy.
- Language reference for the .adm language used by registry-based policy.

After reading this document, you can find out more about Group Policy from the following resources:

- [Windows 2000 Group Policy white paper.](#)
- [Microsoft Platform Software Development Kit home page.](#)
- [Windows 2000 Server Resource Kit Deployment Planning Guide](#) and the [Windows 2000 Server Resource Kit.](#)

Overview of Registry-Based Policy

Registry-based policy is the simplest and most common type of policy setting. This type of policy is implemented using:

- The Administrative Templates extension snap-in in the Group Policy snap-in to configure which policies are applied from the server side.
- A built in registry client side extension on every Windows 2000 or higher client to process the data and create the client registry keys.

Registry-based policy settings are stored in any of the four Group Policy keys listed below. These are considered the approved registry locations for policy settings.

For computer policy settings:

- HKLM\Software\Policies (The preferred location)
- HKLM\Software\Microsoft\Windows\CurrentVersion\Policies

For user policy settings:

- HKCU\Software\Policies (The preferred location)
- HKCU\Software\Microsoft\Windows\CurrentVersion\Policies

These locations have security permissions so that a standard user cannot change these keys to disable or change the behavior of applied policies. The keys are created when the GPO is applied. If the GPO that applied the keys is ever removed, the registry keys associated with it will also be removed at that time.

Note: A local administrator can overwrite these registry keys and thus change or disable the behavior of the policy. (Refer to the [Windows 2000 Group Policy white paper](#) for more information.)

Policy versus Preference

Although Group Policy settings and preferences can both be implemented using registry settings, it is important that you understand the differences between them, when to use which, and when to use them together. Policies and preferences both contain configuration information, and can be used alone or together.

Defining Policies and Preferences

Preferences are set by the user or by the operating system at installation time. The registry values that store preferences are not located under the Group Policy keys listed in the preceding section. Users can typically change their preferences at any time, usually through the user interface of your application. For example users may decide to change the location of their local dictionary for Microsoft Word to a different location or to set their wallpaper to a different bitmap.

Group Policy settings are set by administrators and take precedence over user preferences; these registry values are stored under the approved Group Policy keys. The Group Policy keys are secure, so users cannot change or disable these settings.

Although Group Policy settings take priority over a preference, they *do not* overwrite or touch the registry key used by the preference. If both a policy and preference are present, the preference will be successfully restored if the policy is removed or disabled. Preference settings persist in the registry until they are reversed by a counteracting policy setting or by editing the registry.

How to Use Both Policy Settings and Preferences

It is common practice to offer both a preference and a policy setting for most applications. When building components of your application, you may want to offer the ability to allow a user to configure part of your component and also control this setting centrally using a registry-based policy.

The configuration of the wallpaper on the Windows desktop provides us with an example of where both a policy and preference co-exist. For example, in the Windows shell, it is possible for users to configure their desktop wallpaper to be displayed using the **Display** icon in **Control Panel**. The desktop wallpaper can also be configured using a policy setting that ships in Windows 2000. A policy called **Active Desktop Wallpaper** can be found in the Group Policy snap-in, under the **User Configuration\Administrative Templates\Desktop\Active Desktop** node. Administrators can use this policy to specify the desktop wallpaper that is displayed on users' desktops.

The following table lists the resultant behavior for Group Policy settings and preferences.

Table 2. Results of Group Policy Settings and Preferences.

Scenario	Policy present	Preference present	Resultant behavior
1	No	No	Default
2	No	Yes	Preference configures behavior
3	Yes	No	Policy configures behavior
4	Yes	Yes	Policy configures behavior. Preference is ignored

Note: If you disable or remove a policy, the preference will take effect again. Preference settings are *not* overwritten by any policy setting in the registry, as they use different keys for both the policy and preference.

When to use Registry-Based Group Policy

As stated previously, registry-based policy is simple and easy to implement for the developer. For the administrator, registry based policy is easy to configure and deploy to users.

The following are some questions you can ask to determine whether registry-based policy is the correct choice for your application. Before proceeding, consider the types of things you want to control with Group Policy.

- If you want to create available and non-available type functionality for a part of your component, registry-based policy is an excellent choice as it will give administrators a switch to turn functionality on or off by configuring the policy.

Example:

Let us assume you want to control whether a certain item is displayed or not. You may be able to create a Group Policy that either enables or disables having this item displayed.

- If the type of policy you want to create will define a set of static modes, registry-based policy is also a good choice.

Example:

You would like to create a Group Policy that sets the language that is used by a computer. You have a static list of the selections that an administrator can choose from.

When the administrator enables the policy setting from the Administrative Templates snap-in, he or she will be provided with a list of languages to choose from.

- If the type of policy you want to create requires simple input, which can be stored in the registry as plain text from the administrator to configure the policy, then registry-based policy is a good choice.

Example:

- You would like to create a Group Policy to define the screensaver or bitmap to be displayed.
- When users enable this policy setting, they are given a text dialog to enter the name and path of the file to be used. This information is stored to the registry as plain text.
- Your application checks for this information and behaves accordingly.

-
- If the policy you want to create can be configured using a simple UI and this configuration information can be stored in the registry as plain text, you should consider registry-based policy.
The UI controls provided by the Administrative Templates snap-in in which your registry-based policy will be stored are:

- CheckBox
- ComboBox
- DropDownList
- EditText
- ListBox
- Numeric with Optional SpinControl
- Static text

These controls are described in the .adm language section of this document.

Designing and Developing Registry-Based Policy

Design Considerations

It is important to clearly define what aspects of your component or application you would like to enable with Group Policy.

For example, consider the following issues:

- What specific things would you like policy to control?
- How many policy settings will it require to control this behavior? Refer to the section on [Best Practices for User Interface Design](#) for more information.
- What is the default behavior? What is the behavior when the policy is enabled or disabled? How is this reflected in the UI?
- Will the policy settings affect users or computers or possibly both?
- Understand the UI capabilities that the .adm language offers. The .adm language will be used to build the UI that is presented to administrators when they configure the policy using the Group Policy snap-in.

Best Practices for When to Create Group Policy

This section highlights best practices for when it would be appropriate to create Group Policy.

- **Policy on/off.** Provide a single policy setting that controls whether or not policy is to be applied to your application at all. It should handle allowing defaults and/or tightly managed type policy.
For example:
HKCU\Software\Policies\CoName\AppNameVersion\Settings, with the value of "Policy" the data would be one of the following values: 0,1,2,3.
With 0 = No policy applied; 1 = only apply defaults; 2 = only apply mandated policy; and 3 = apply both.
This allows the administrator to have a 'switch' that can be used to control the behavior of the application.
- **New options for a new release.** Provide policy settings for all of the new features of the application. Provide a single Group Policy for all of the new features, as well as a policy for each logical grouping of new features. A policy should also be considered for specific features that administrators would need to control after the new features have been enabled.
Rationale - One significant cause for upgrade delays in corporate environments is the administrator's inability to rollout the new version of an application without overwhelming users and the support staff with the new features. Enabling policy settings in this area will allow the administrator to control how and when users get the features. Grouping related features will allow the administrator to prevent use (or lock-out) of a new feature set until users have been trained.

-
- **Options that create support problems.** Determine the top issues that users have with the application and consider ways that policies could be used to prevent the support call. This applies to both the applications support and also to “in-house” corporate support centers.
 - **Feature re-design.** When a product feature is being redesigned, consider why and if a policy could help address the reason for the re-design. For instance, if the feature was deemed overly complex, consider creating a Group Policy that would shield users from the more advanced functions.
 - **Requires advanced knowledge of the application.** If there are complicated or more advanced settings that the typical user does not normally need to know about, use Group Policy to give the administrator the ability to control access to the settings.

Data population. Use policy to populate data for your application. For example, a phone dialer could use policy to provide the administrator with the ability to either default and/or mandate certain items in the phone directory.

- **Home market issues.** Group Policy settings are not just for the corporate environment. Consider issues like limiting functions for different user skills or feature levels for home users sharing the same computer (dad, mom, 3 to 10 year olds, 11 to 21 year olds, and so on).

- Do you need or already support preferences?

Can the item that you need to policy-enable be set by a preference? If so, you will want to ensure that you have covered the scenarios of having both the Group Policy and preference in existence. Refer to the [How to Use Both Policy Settings and Preferences](#) section, earlier in this document.

Define a registry key and values for these settings. Any registry keys that are required for your policy settings must be located under the approved Group Policy keys. If your policy is based on an existing preference, then use the same registry key/value under the policies key.

- After you understand how your policy will behave, make sure to provide details on how this will be reflected in the user interface. The user interface must always adhere to policy settings that are applied. Policy settings should have a visible, immediate effect on the UI. If possible, the policy settings should hide the UI that is associated with this policy.

Best Practices for User Interface Design

This section highlights best practices for user interface (UI) issues related to creating Group Policy.

The following example lists some design options that developers could use to disable **My Network Places**:

- Do nothing. In this case, the user clicks **My Network Places** and nothing is displayed. The user will assume something is wrong and call the helpdesk. This would be a poor design option. Correctly creating and using Group Policy should reduce IT calls not increase them.
- Create an error message. In this case, the user clicks **My Network Places** and an error message is displayed saying: "This has been disabled by your administrator." The user will call the administrator to ask why this feature has been disabled. Again this is not recommended.
- Disable the UI. A disabled (grayed-out) UI feature typically implies that there is a way to enable the UI. The user will likely spend a lot of time trying to get this feature to work. In the end, they will either give up frustrated or call the helpdesk.
- Hide the UI feature. If you choose to hide the UI, the user will generally not recognize that anything is wrong. This is the preferred choice for Group Policy settings.

Creating Explain Text for Group Policy

Developers and technical writers typically work together to write the Explain text that is displayed in the UI whenever an administrator tries to configure your policy setting. This text is an important piece of documentation that should explain the behavior of your policy, its interaction with other policies, and any other notes that you would like to make users aware of. More details and a template are provided later in this document in the [Writers Role in developing Registry-Based Policy](#) section.

[.adm Files](#)

To determine which registry-based policies are currently available for the operating system, you can read the .adm files provided with Windows 2000 Server. To do this, you can use a text editor such as Notepad. The .adm files that ship with Windows 2000 Server are located in the folder: %windir%\inf\

Note: You should treat the .adm files that ship in the operating system as read only. These files will be updated from time to time in service packs and in the future releases of the product. If you want to customize these files, you should make a copy and save them using a different filename.

You will want to determine if there are any policies similar to the ones you would like to create. Looking for similar policies will provide the following:

-
- A model to use as a basis for your Group Policy settings.
 - Consistency with available policy settings.
 - Information about possible interactions between policies.

When you create the specification document for your component, you should make sure to include all this type of information.

Best Practices for Developing Registry Based Policy Settings

For the design of your policy, consider the following issues:

- Try not to make a single policy to control all aspects of your component. Group Policy is easier to implement and use if you have several smaller policies. This approach gives the user more flexibility.
- Your policy should have associated behavior for each of the following three possible states:
 - **Enabled** should turn on the behavior indicated by the policy name.
 - **Disabled** should turn on the default behavior.
 - **Not Configured** should have no effect.
- The User Interface should always reflect the policy applied.
- You should aim to have an associated policy for each user preference in your application.

When to Create a Policy

You should create a policy for the following purposes:

- To hide or disable new behavior that may confuse your customers.
- To hide settings and options that users may waste time setting and unsetting.
- To hide or disable a UI that can lead users into a situation in which they must call the help desk for support.
- To help customers administer their computers and lock-down their desktops.

When Not to Create a Policy

You should not create a policy:

- For all your application settings—be selective about the features you would like to enable or disable.
- If you do not intend to provide support for the policy setting. Treat each policy as a feature that needs to be tested, validated, and supported.

Writer's Role in Developing Registry-Based Group Policy

The writer plays two important roles in developing registry-based policy settings: creating the policy name and writing the Explain text. Both are detailed here.

Creating Names for Group Policy

The writer's first task is to help develop the name of the policy setting that will be displayed in the Group Policy snap-in. The Group Policy name should be short but also reflect what the policy does.

Note: Policy names are limited to 256 characters. However, depending on the font used on the user's workstation, a smaller number of characters will be displayed, and all others truncated. On most systems, you can assume that you will have the ability to display policy names up to 65 characters. (Resolution 1024 / 768 with small fonts installed).

Best Practices for Naming Group Policy

- When Group Policy names get localized to foreign languages, they typically require additional characters. Therefore, we recommend that if you are using English to name your Group Policy, you limit it to 49 characters. (This allows your title to grow by 33 percent during localization without causing any truncation issues when displayed.)
- Do not try to supplement your Group Policy title in the UI with a TIP text that can be displayed in the UI using the **TEXT Part Type** provided by the .adm language.

Creating the Explain Text

The writer's second task in developing registry-based policy is creating the associated Explain text for each policy setting. Explain text is displayed in the policy **Properties** dialog box whenever an administrator selects a policy setting, and then clicks the **Explain** tab. This text is an important piece of documentation that provides information about the behavior of the policy, its interaction with other policies, and any other issues that you would like to make administrators aware of.

It is helpful to draft the Explain text soon after creating the specification document. This will serve as a high-level roadmap for developers, and it also assists testers in creating a test plan for the policy.

Note: Explain text is limited to a maximum of 4096 characters.

The following is a template that you may want to use to layout the Explain text. You will want to include the following items (listed in order):

1. A one or two line description of the policy.
2. A one or two line description of the feature that the policy affects.

3. A description of the behavior of the policy when it is enabled.
4. A description of the behavior of the policy when it is disabled.
5. A description of the behavior when the policy is not configured.
6. Any tips for using the policy setting.
7. Any notes or interactions that this policy has with other settings.

As a **Best Practice**, you should provide information on:

- Items that are not covered by this policy setting.
- Any other policies required for your policy to function.
- Any other policies that are related to the same component that your policy affects and which may have a higher or lower priority. An example of this would be if you have a policy to restrict access to the LAN settings for a computer. This policy would take priority over any more granular policy settings that covered the actual items you can configure within a LAN connection.
- Any related policies that you would like to make the administrator aware of.

Sample Explain Text

The following is an example of **Explain** text from the System.adm file that is included in Windows 2000 Server:

The policy displayed is: **Administrative Templates\User Settings\Desktop\Active Desktop\Active Desktop Wallpaper.**

Specifies the desktop background ("wallpaper") displayed on all users' desktops.

This policy lets you specify the wallpaper on users' desktops and prevents users from changing the image or its presentation. The wallpaper you specify can be stored in a bitmap (*.bmp), JPEG (*.jpg), or HTML (*.htm, *.html) file.

To use this policy, type the fully-qualified path and name of the file that stores the wallpaper image. You can type a local path, such as C:\Winnt\Logo.bmp or a UNC path, such as \\Server\Share\Logo.bmp.

If the specified file is not available when the user logs on, no wallpaper is displayed. Users cannot specify alternate wallpaper.

You can also use this policy to specify that the wallpaper image be centered, tiled, or stretched. Users cannot change this specification.

If you disable this policy or do not configure it, no wallpaper is displayed. However, users can select the wallpaper of their choice.

Note: This policy requires that Active Desktop be enabled. By default, Active Desktop is disabled. To use a policy to enable Active Desktop, use the "Enable Active Desktop" policy.

Also, see the "Allow only bitmapped wallpaper" and the "Disable changing wallpaper" policies.

Developers Role in Developing Registry-Based Policy

- 1) Understand how and when policy is applied.
- 2) Understand the behavior of the policy to be developed. When deploying your policy, an administrator will be able to set your policy settings to one of three states: Enabled, Disabled or Not Configured. The specification document should detail the behavior for all three of these states.
- 3) Select registry location, naming and data types.
 - Registry keys used for your policy setting must live in one of the policy keys:
 - i. HKCU\Software\Policies (preferred location)
 - ii. HKLM\Software\Policies (preferred location)
 - iii. HKCU\Software\Microsoft\Windows\CurrentVersion\Policies
 - iv. HKLM\Software\Microsoft\Windows\CurrentVersion\Policies
 - The folder in which your registry keys are created under these policy keys is up to you. We recommend that you mimic the directory structure of any associated preferences that you have under the policies key that you suggest.

For example, if your component stores its preferences in the location: `Software\Microsoft\Windows\Component Name`

Then you would store the associated policy settings in the location:

`Software\Policies\Microsoft\Windows\Component Name`

- 4) Select values that can be assigned to these registry keys that you selected in Step 2.

You will want to choose an appropriate registry value for each of these states that your code will look for under the key you selected in Step 2.

- The Enabled and Disabled states should have an associated registry key and value.
- The Not Configured state should not write any value to the registry.
- For the actual keys that will be used, the only types of data that can be stored to them are:
 - i. REG_DWORD
 - ii. REG_SZ
 - iii. REG_EXPAND_SZ
- Confirm the values selected with the developer for each of these three states.
 - a. You will want to confirm these values with the developer or the writer of the .adm file so they match.
 - b. These values should be documented in the specifications.

- 5) Modify your component to check the policy key for the registry key and associated value(s) that you selected to be used for your policy. In the case where you have a policy and preference:

- Read the policy value first, and then if not found, read the preference. (Policy always takes priority)
- Use the standard registry functions to read both the policy and the preference.
- If no policy exists, then default in the code. In most instances you will read an associated preference at this point if one exists.

- Consider the sample code below to perform this lookup for you.

```
#define PREFERENCE_KEY
TEXT("Software\\Microsoft\\Windows\\CurrentVersion\\
Explorer")
#define POLICY_KEY
TEXT("Software\\Policies\\Microsoft\\Windows\\Explor
er")

DWORD ReadValue (LPTSTR lpValueName, DWORD
dwDefault)
{
    HKEY hKey;
    LONG lResult;
    DWORD dwValue, dwSize = sizeof(dwValue);

    // First, check for a policy
    lResult = RegOpenKeyEx (HKEY_CURRENT_USER,
POLICY_KEY, 0,
                                KEY_READ, &hKey);
    if (lResult == ERROR_SUCCESS)
    {
        lResult = RegQueryValueEx (hKey,
lpValueName, 0, &dwType,
                                (LPBYTE)
&dwValue, &dwSize);
        RegCloseKey (hKey);
    }

    // Exit now if a policy value was found
    if (lResult == ERROR_SUCCESS)
    {
        return dwValue;
    }

    // Second, check for a preference
    lResult = RegOpenKeyEx (HKEY_CURRENT_USER,
PREFERENCE_KEY, 0,
                                KEY_READ, &hKey);
    if (lResult == ERROR_SUCCESS)
    {
        dwSize = lResult = RegQueryValueEx (hKey,
lpValueName, 0,
                                &dwType, (LPBYTE)
&dwValue, &dwSize);
        RegCloseKey (hKey);
    }
}
```

```
// Exit now if a preference was found
if (lResult == ERROR_SUCCESS)
{
    return dwValue;
}

// Neither a policy or a preference was found,
so return the default value
return dwDefault;
}
```

- 6) Make sure that the UI for your component adheres to any policies that you create. If your policy removes or disables functionality this must be reflected in the UI. Refer to more details in the [Developers Role in Developing Registry-Based Policy](#) for UI behavior guidelines.
- 7) The component which you are policy enabling should check the appropriate policy keys when your component starts and also during a policy refresh. Policy is refreshed periodically and the assigned policy settings could change. Therefore, if you have a policy setting that changes the user interface, it is even more important for you to monitor when a policy refresh occurs so you can refresh the display.

For example:

- You create a policy that removes the Find command from a menu in your application. When this policy is enabled, a registry key is set to indicate that the Find command should not be available.
- When your application starts it should check for this key, and if so make the Find command unavailable.
- But what if the user is already logged on, and has the application open? In this scenario, when policy is refreshed, you want this change to take effect immediately and not require the user to close and reopen the application. By watching for a policy refresh, and refreshing the display, you can prevent this problem.

You can implement this using two different methods. The first is to use the `RegisterGPNotification` API to allow you to be notified when Group Policy has been changed. To use `RegisterGPNotification`, an application will more than likely do this from a background thread. A code sample is provided below:

```
//
// Call CreateThread to watch for GP notifications
```

```

//

hThread = CreateThread (NULL, 0, (LPTHREAD_START_ROUTINE)
NotifyThread,
                        0, 0, &dwID);

// GP notification thread. When the events are signalled, a
message is added to the main window

DWORD NotifyThread (DWORD dwDummy)
{
    HANDLE hHandles[3];
    DWORD dwResult;

    RegisterGPNotification(hMachineEvent, TRUE);
    RegisterGPNotification(hUserEvent, FALSE);

    hHandles[0] = hExit;
    hHandles[1] = hMachineEvent;
    hHandles[2] = hUserEvent;

    while (TRUE) {
        dwResult = WaitForMultipleObjects (3, hHandles, FALSE,
INFINITE);
        if ((dwResult == WAIT_FAILED) || ((dwResult -
WAIT_OBJECT_0) == 0)) {
            if (dwResult == WAIT_FAILED) {
                AddString (TEXT("WaitForMultipleObjects
failed."));
            }
            break;
        }

        if ((dwResult - WAIT_OBJECT_0) == 1) {
            AddString (TEXT("Machine notify event signaled."));
        } else {
            AddString (TEXT("User notify event signaled."));
        }
    }
    UnregisterGPNotification(hMachineEvent);
    UnregisterGPNotification(hUserEvent);

    return 0;
}

```

The second method to implement this is by watching the WM_SETTINGCHANGE window message with the IParam set to "Policy". A code sample is provided below:

```

case WM_SETTINGCHANGE:

    if (!strcmpi ((LPTSTR)lParam, TEXT("Policy"))) {

        if (wParam) {
            AddString (TEXT("Received WM_WININICHANGE:
machine policy applied."));
        }
    }
}

```

```
        } else {
            AddString (TEXT("Received WM_WININICHANGE:
user policy applied."));
        }
    }
    break;
```

Best Practice for Developing Registry-Based Policy:

- If security is important to you, you should try to implement registry-based policy at the lowest level possible. If you can make your APIs policy aware, it will be very difficult for the policy to be defeated. If you implement the policy at a higher level, possibly just at the UI level, a user can create an application that will bypass your policy.
- How deep you need to go is determined by what you want to provide to your customer. Most times, you will use policy to simplify the user experience and reduce the TCO of the desktop. In this case you can get away with implementing policy settings at a higher level in your code.

Testing Registry-Based Policy

Create a test plan. Your test plan should clearly document:

- 1) What is the default behavior? What is the behavior when the policy is enabled, disabled, and not configured?
- 2) What are the possible settings that the policy can be configured as? What is the associated behavior?
- 3) Is there an associated preference? What is the behavior of the preference?
- 4) What changes in the UI when a policy is enabled, disabled, or not configured?
- 5) If you allow the administrator to give input when setting a policy, you should test the behavior for incorrect input.
 - For example: You create a policy to configure the bitmap to be displayed when your application, and the policy setting requires the administrator to enter a path to the bitmap. You will want to test what occurs if the path is incorrect or if the file is not present.
- 6) Test your new policies individually first. Then test how each policy interacts with other policies that are similar, or affecting the component that your new policy affects.

Example: You create a policy to configure the wallpaper that will be displayed on your clients' desktops. If you check the current list of policy settings that ship with Windows 2000, you will find that there are other wallpaper policies that already exist. In this scenario, you

should test how these policies interact. Any issues that arise need to be addressed or documented in the Explain text.

Best Practice:

Testing should ensure that the UI is policy aware. If the UI is not aware of the policy, the user experience will be confusing and cause confusion for the end user.

For example: If your policy restricts access to a certain item in your component, then all access to this component and its configuration should not be available in the UI. Some of the possible ways to achieve this are:

- a. Removing the item completely visually.
- b. Grey-out the item and disable it.

Building Your [.adm](#) File.

Once you have modified your component to check the appropriate registry keys and behave accordingly, you need to create an .adm file.

The .adm file will be used to configure your Group Policy settings on the client computers.

Once you have created an .adm file, copy it into the Group Policy snap-in. Using the Group Policy snap-in, administrators can configure and deploy the policy registry settings to client computers.

For the details on how to create an .adm file, please refer to [Appendix A: .adm Language Reference](#).

Loading your [.adm](#) File Into the Group Policy Snap-In

Once you have created an .adm file to test your policy, you can load this template into the Administrative Templates extension snap-in by performing the following steps:

- 1) Load the Group Policy snap-in.
 - At a command prompt type: **GPEDIT.msc** and press **ENTER**.
- 2) Under either Computer Settings OR User Settings, right click on **Administrative Templates**.
- 3) On the context menu that appears, click on **Add/Remove Templates**.
- 4) A new dialog box will appear that will allow you to add or remove .adm templates. Click on **Add**.
- 5) Enter the name of the filename of the .adm file that you would like to add.
- 6) Click on **Open**.

-
- 7) If your .adm file was successfully loaded, you will be returned to the dialog that you saw in Step 4. In this case click on **Close**. Your policy template has been added successfully. Skip all the steps below.
 - 8) If your .adm file was not successfully loaded, you will be presented with a dialog displaying the errors that occurred during the loading of .adm.
 - At this point, make a note of the errors that were found. Click on **OK**.
 - You will be returned to the dialog that you saw during Step 4. Although your .adm file was not successfully loaded, it will still appear in the list of .adm files loaded.
 - Select your .adm file, and click on **Remove**.
 - Click on **Close**.
 - You are now back to the Group Policy snap-in. At this point, edit your .adm file and correct any problems. Then repeat this process again starting from Step 2, to try to load your .adm template again.

Appendix A: [.adm](#) Language Reference

Once you have policy enabled your application to use registry-based policy, you need to create a method for a local administrator or domain administrator to enable and configure your policy setting. All registry-based policy settings appear and are configured using the Administrative Templates snap-in in the Group Policy Editor (GPE).

You must create a text file that describes your policy settings using the .adm language for your policy to be available in the GPE. The .adm language provides a framework language.

Once an .adm file is created to detail your policies, an administrator can add this template to the Administrative Templates snap-in and the policies will appear in the UI. Multiple .adm files can be loaded in the Administrative Templates snap-in at the same time. By default with Windows 2000 or higher, the following .adm files are already loaded:

- System.adm
- Conf.adm
- Inetres.adm

Sample [.adm](#) File

For an example of a completed .adm file, see Figure 1 below. The following sections of this document will provide more specific details of the syntax and functionality available in the .adm language.

The .adm code is presented on the left, and a screenshot on the right reflects the effect of this template in the Group Policy snap-in.

```
CLASS USER
```

```
CATEGORY !!DesktopLockDown
```

```
KEYNAME "Software\Policies\System"
```

```
POLICY !!DisableTaskMgr
```

```
EXPLAIN !!DisableTaskMgr_Explain
```

```
VALUENAME "DisableTaskMgr"
```

```
VALUEON NUMERIC 1
```

```
VALUEOFF NUMERIC 0
```

```
END POLICY
```

```
[strings]
```

```
DisableTaskMgr="Disable Task Manager"
```

```
DisableTaskMgr_Explain="Prevents users from starting Task Manager"
```

```
DesktopLockDown="Desktop Settings"
```

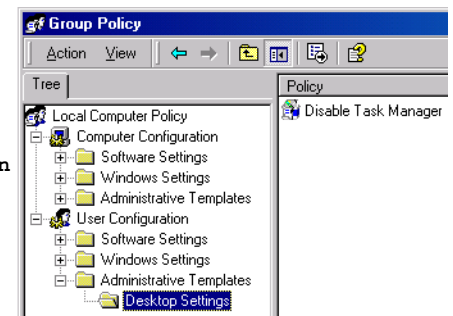


Figure 1. Example of .adm code and results in Group Policy.

This sample .adm code allows you to configure a policy called "Disable Task

Manager,” which appears in the Group Policy Editor namespace: User Configuration\Administrative Templates\Desktop Settings.

This policy achieves the following:

- If you enable this policy it will create a registry key called DisableTasMgr and set its value to 1.
- If you disable this policy it will create a registry key called DisableTasMgr and set its value to 0.
- In both cases the DisableTasMgr key will be created below HKCU\Software\Policies\System

This sample gives you a very basic example of what you can do with .adm files. The following sections of this document walk you through more details of the .adm language.

Components of the [.adm](#) Language:

Each .adm file can contain zero or more policies, and each policy in turn can contain zero or more parts. The .adm language includes the following components:

- [Comments](#)
- [Strings](#)
- [CLASS](#)
- [CATEGORY](#)
- [POLICY](#)
- [PART](#)
- [ITEMLIST](#)
- [ACTIONLIST](#)

Comments

There are two methods that you may use to add comments to an .adm file. You can precede the comment either with a semicolon (;) or two forward slashes (//). Or you can place comments at the end of any valid line.

Strings

To add strings to an .adm file, precede the text with two exclamation points (!!). At the end of your .adm file, all strings must be defined in the **[strings]** section of the .adm file. The strings must be enclosed in quotes. Optionally, you can enclose a variable name or hard-coded string in double quotation marks.

Example:

```
POLICY !!LimitSize
    EXPLAIN !!LimitSize_Explain      ; This string is stored in the
strings section
    TIP1 "Limit Profile Size to"     ; This string is hard coded

[strings]
LimitSize="Limit profile size"
LimitSize_Explain="Limits the size of user profiles"
```

Best Practice:

It is a best practice to place all strings used in your .adm file in the **[strings]** section of the .adm file. This facilitates conversion of the .adm file to other languages (localization), as you will only need to modify the [strings] section of an .adm file to port it to different languages. Any names and strings with spaces in them must be enclosed in double quotation marks.

CLASS

This component is used to define where your policy will appear in the Group Policy Editor snap-in.

The first entry in the .adm file is the keyword **CLASS**. We use this to specify whether the subsequent entries should be displayed under the **Computer Settings** or the **User Settings** node of the Group Policy snap-in.

Syntax

The CLASS syntax is as follows:

```
CLASS name
```

name

This defines the name of the Class, which must be **MACHINE** or **USER**. If the .adm file contains a Class other than the valid Classes (MACHINE or USER), the errors are ignored when the Group Policy snap-in loads.

CLASS Example

The following example illustrates use of **CLASS**.

```
CLASS MACHINE
```

```
CLASS USER
```

Note: You can define multiple CLASS USER or CLASS COMPUTER sections in an .adm file. When the file is processed, all the CLASS USER sections will be merged. The same occurs for all CLASS COMPUTER sections.

CATEGORY

Once you have defined if your policy will appear under the **Computer Settings** or **User Settings** node of the Group Policy snap-in using the **CLASS** component, you may need to use the **CATEGORY** component to display a node name under which your policy setting will be displayed in the Group Policy snap-in.

Note: To create child nodes, you may nest CATEGORY within CATEGORY.

Syntax

The **CATEGORY** syntax is as follows:

```
CATEGORY !!name
           KEYNAME key name
           [policy definition statements]
END CATEGORY
```

name

The **CATEGORY** name as it should appear in the Group Policy snap-in list box. Optionally, you can enclose the variable name in double quotation marks. Names with spaces must be enclosed by double quotation marks.

key name

This is an optional path to the registry key to use for the **CATEGORY**. Do not use **HKEY_LOCAL_MACHINE** or **HKEY_CURRENT_USER** in the registry path; the preceding **CLASS** statement specifies which of these keys to use.

If you specify a key name, all child categories, policies, and parts will use this key name, unless they specifically provide a key name of their own. Names with spaces must be enclosed in double quotation marks.

policy definition statements

Categories can include zero or more **POLICY** statements. A policy definition statement cannot appear more than once within a single category.

CATEGORY Example

The example in Figure 2 below illustrates the use of **CATEGORY** and nesting.

```
CLASS USER
```

```
; The following categories  
will be displayed  
; under User settings
```

```
CATEGORY !!Desktop
```

```
KEYNAME  
"Software\Policies\System"  
  
; <INSERT POLICIES HERE>
```

```
CATEGORY !!InternalApps
```

```
KEYNAME  
"Software\Policies\InternalApp  
s"  
  
; <INSERT POLICIES  
HERE>
```

```
END CATEGORY  
END CATEGORY
```

```
[strings]
```

```
Desktop="Desktop Settings"
```

```
InternalApps="Line of Business Apps settings"
```

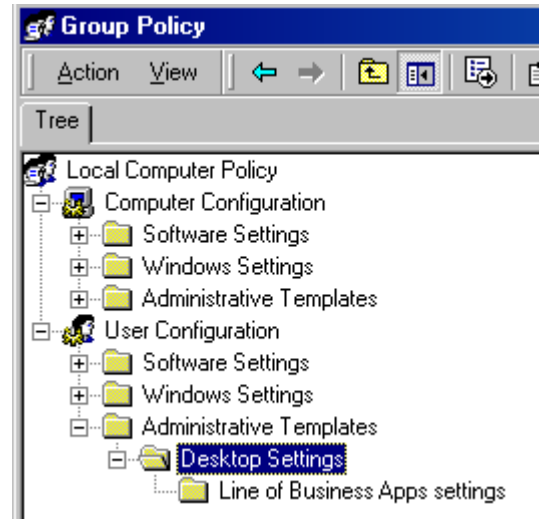


Figure 2. Example of CATEGORY.

Keywords

The valid keywords for **CATEGORY** are:

- **KEYNAME**
- **CATEGORY**
- **POLICY**
- **END**

Note: If you have a **CATEGORY** defined with a default **KEYNAME** in it, and the same category is found again later in the .adm file, that same default **KEYNAME** is still in effect. This means it is possible that you could get an error message about **KEYNAME** being defined twice, when it was actually just defined in the same category earlier.

POLICY

To identify a policy that the user can modify, you use the keyword **POLICY**. The policy and its associated controls are displayed in a dialog box that administrators use to set the state of the policy. You can use multiple **POLICY** key names under one **KEYNAME**.

The following examples in Figure 3 illustrate the syntax of **POLICY**.

Syntax

```
POLICY name
    [KEYNAME key name]
    EXPLAIN help string
    VALUENAME value name
    CLIENTTEXT guid
    [part definition statements]
END POLICY
```

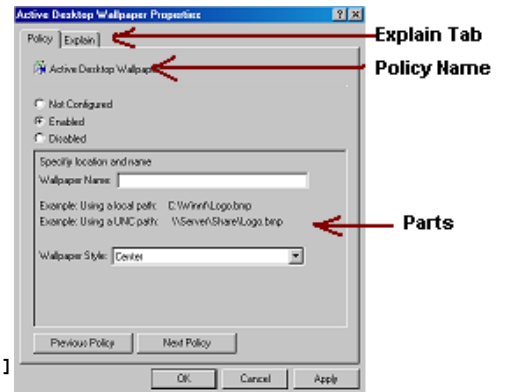


Figure 3. Example of **POLICY**.

name

The name of the policy as it should be displayed in the Group Policy snap-in namespace.

key name

This is an optional path to the registry key to use for the category. Do not include **HKEY_LOCAL_MACHINE** or **HKEY_CURRENT_USER** in the registry path; the preceding **CLASS** statement determines which of these keys is used.

If you specify a key name, all **PART** definition statements will use this key name unless they specifically provide a key name of their own.

help string

This is the text string that is displayed in the **Explain** tab of the policy's dialog box.

value name

This is the registry value to modify. Selecting the option sets the value as a REG_DWORD of 1. Clearing the option removes the registry value. To specify values other than the default values, use the **VALUEON** and **VALUEOFF** statements directly following the corresponding **VALUENAME** statement. These statements are specified as follows:

VALUEON *on value*

VALUEOFF *off value*

When you use these statements, the behavior is modified such that if the administrator selects the option, the value is set to *on value*. If the administrator clears the option, the value is set to *off value*.

guid

This is an optional value that specifies the GUID of the snap-in extension.

part definition statements

A policy can contain zero or more **PART** statements, which specify various options, including drop-down list boxes, text boxes, and text in the lower pane of the Group Policy snap-in.

POLICY Example

The following example in Figure 4 illustrates the use of **POLICY**.

```
CLASS MACHINE

CATEGORY !!DiskQuota

    KEYNAME "Software\Policies\MS\DiskQuota"

    POLICY !!DQ_Enable
        EXPLAIN !!DQ_Enable_Help
        VALUENAME "Enable"
        VALUEON NUMERIC 1
        VALUEOFF NUMERIC 0
        CLIENTTEXT {3610eda5-77ef-11d2-8dc}

        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"
```

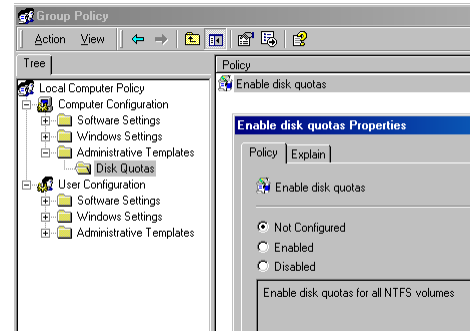


Figure 4. Example of using the **POLICY** component for Group Policy.

KEYWORDS

The valid keywords for **POLICY** are:

- **KEYNAME**
- **PART**
- **VALUENAME**
- **VALUEON**
- **VALUEOFF**
- **ACTIONLISTON**
- **ACTIONLISTOFF**
- **END**
- **HELP**
- **CLIENTTEXT**
- **POLICY**

PART

You can use **PART** to specify various options, such as drop-down list boxes, text boxes, and text in the lower pane of the Group Policy snap-in. See Figure 5 below.

Note:

For a simple policy where you only need to set a registry key to either 1 or 0, you will not need to use PART. PART allow you to make the system administrator experience richer and also collect more information from the administrator through simple controls.

Syntax

The **PART** syntax is:

PART name *part-type*

type-dependent data
[**KEYNAME** key name]
VALUENAME value name

END PART

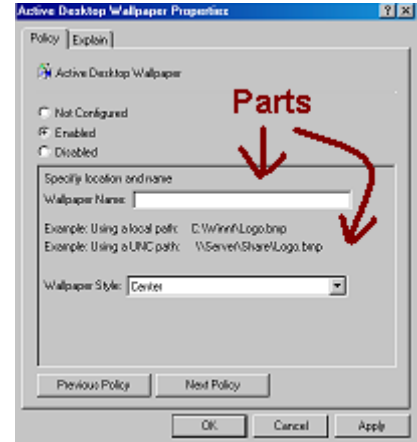


Figure 5. Example of PART.

name

Specifies the **PART** name as it should appear in the Group Policy snap-in. It may optionally be enclosed by double quotation marks. Names with spaces must be enclosed by double quotation marks.

part-type

- A policy **PART** type. This can be one of the types shown in the following table.

Table 3. Policy PART Types.

CHECKBOX	Displays a check box. The value is set in the registry with the REG_DWORD type. The value is other than zero if the check box is checked, and zero if it is not checked.
COMBOBOX	Displays a combo box.
DROPDOWNLIST	Displays a combo box with a drop-down list style. The user may choose only one of the entries supplied.
LISTBOX	Displays a list box with Add and Remove buttons. This is the only PART type that can be used to manage multiple values under one key.

EDITTEXT	Displays a text box that accepts alphanumeric text. The text is set in the registry with either the REG_SZ or the REG_EXPAND_SZ type.
TEXT	Displays a line of static text. There is no associated registry value with this PART type.
NUMERIC	Displays a text box with an optional spin control that accepts a numeric value. The value is set in the registry with the REG_DWORD type.

type-dependent data

This is information about the **PART**.

key name

This is an optional path to the registry key to use. Do not include **HKEY_LOCAL_MACHINE** or **HKEY_CURRENT_USER** in the registry path. If no key name is specified, the previous key name in the hierarchy is used.

value name

Indicates the registry value to modify. Selecting this option sets the value to a **REG_DWORD** of 1, and clearing the option removes the registry value. If you want to specify values other than the default values, use the **VALUEON** and **VALUEOFF** statements directly following the corresponding **VALUENAME** statement. You specify these statements as follows:

VALUEON *on value*

VALUEOFF *off value*

Figure 6 below illustrates the different **PART** types:

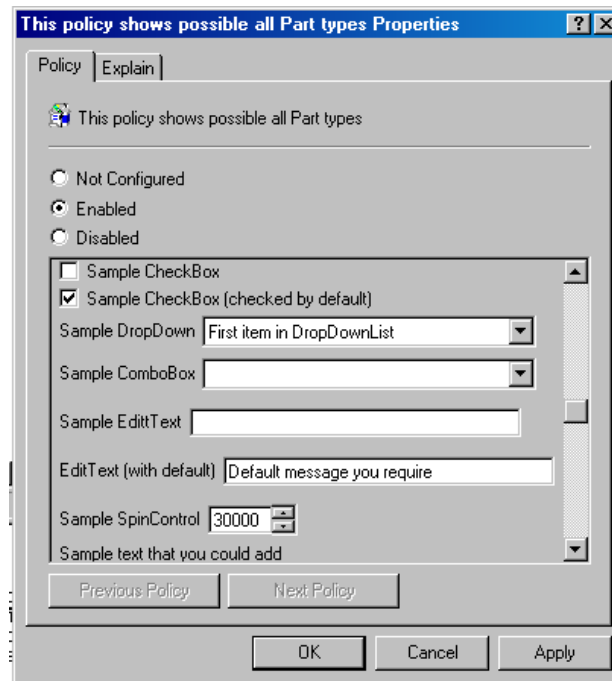


Figure 6. Example of different **PART** types.

Keywords

The valid keywords for **PART** are:

- **CHECKBOX**
- **TEXT**
- **EDITTEXT**
- **NUMERIC**
- **COMBOBOX**
- **DROPDOWNLIST**
- **LISTBOX**
- **END**
- **CLIENTEXT**
- **PART**

Using **PART** Types to Add Controls to Your User Interface.

Using the valid keywords along with the **PART** component allow you to add text and various User Interface controls to the properties page belonging to the policy that you are creating when it appears in the Group Policy editor.

Because much of the syntax is related, the next section will present a task-based approach to writing the syntax for these **PART** types used to create the UI elements above.

Using the different **PART** types, you can add different kinds of text and controls to enhance your policy setting. All of these types need to be used with the **PART** component defined earlier in this document.

Adding Text to Be Displayed on the Property Page of a Policy

1)TEXT

- The **PART** type **TEXT** can be used to display text.

Syntax

```
PART text TEXT
END PART
```

text:

Text to be displayed is entered here. It can be hard-coded and placed with quotes or you can make the string a variable by putting **!!** in front of the variable name.

The following example illustrates the use of **TEXT**. The **Disable Active Desktop** policy deactivates Active Desktop and prevents users from enabling or disabling Active Desktop, or modifying the configuration.

```
POLICY !!NoActiveDesktop
  KEYNAME "Software\Microsoft\Windows\CurrentVersion\Policies\Explorer"
  EXPLAIN !!NoActiveDesktop_Help
  VALUENAME "NoActiveDesktop"

  PART !!NoActiveDesktop_Tip      TEXT
  END PART

END POLICY
```

The valid keyword for **TEXT** is **END**.

Taking System Administrator Text Input from the Property Page of a Policy

2) EDITTEXT

Allows the user to input alphanumeric text in an edit field. The text is set in the registry with the **REG_SZ** type.

Syntax

```
PART text EDITTEXT  
    VALUENAME value name  
END PART
```

text:

Text to be displayed is entered here. It can be hard-coded and placed with quotes or you can make the string a variable by putting **!!** in front of the variable name. This text will be displayed on the left side of the edit box.

value name

This indicates the registry value to which the users input entered in the Edit Text box will be written.

The **PART** type, **EDITTEXT**, accepts the options shown in the following table.

Table 4. Options for EDITTEXT.

DEFAULT <i>value</i>	Specifies the initial string to place in the edit field. If this option is not specified, the field is initially empty.
MAXLEN <i>value</i>	Specifies the maximum length of a string. The string in the edit field is limited to this length.
REQUIRED	Specifies that the Group Policy snap-in does not allow a policy containing this PART to be enabled, unless a value has been entered for this PART .
OEMCONVERT	Sets the ES_OEMCONVERT style in the edit field so that typed text is mapped from ASCII to OEM and back. ES_OEMCONVERT converts text entered in the edit control. The text is converted from the Windows character set (ASCII) to the OEM character set and then back to the Windows set. This ensures proper character conversion when the application calls the CharToOem <code><JavaScript:hhobj_1.Click()></code> function to convert an ASCII string in the edit control to OEM characters. This style is most useful for edit controls that contain file names.

The valid keywords for **EDITTEXT** are:

- **KEYNAME**
- **VALUENAME**
- **DEFAULT**
- **REQUIRED**
- **MAXLENGTH**
- **OEMCONVERT**
- **END**
- **EXPANDABLETEXT**
- **CLIENTEXT**

PART example with EDITTEXT and TEXT

The following example in Figure 7 illustrates the use of the **EDITTEXT** and **TEXT** **PART** types.

```
CLASS USER
CATEGORY !!DesktopLockDown

KEYNAME "Software\Policies\System"
POLICY !!Wallpaper
EXPLAIN !!Wallpaper_Explain

PART !!Wallpaper_Tip1 TEXT
END PART

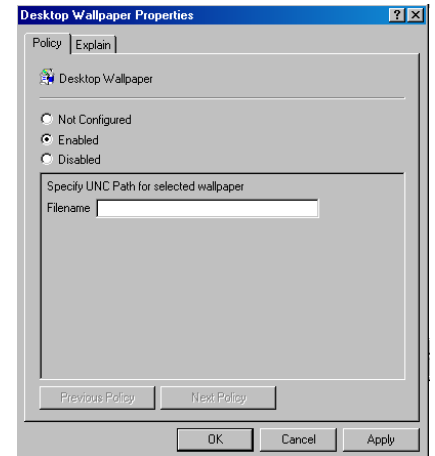
PART !!Wallpaper_Filename EDITTEXT
VALUENAME Wallpaper
MAXLEN 60
END PART

END POLICY

END CATEGORY
```

```
[strings]
DesktopLockDown="Desktop Settings"
Wallpaper="Desktop Wallpaper"
Wallpaper_Explain="Used to set the desktop wallpaper"
Wallpaper_FileName="Filename"
Wallpaper_Tip1="Specify UNC Path for selected wallpaper"
```

Figure 7. Example of **EDITTEXT** and **TEXT** **PART** types.



In this example, when the policy setting is enabled the text entered into the edit field will be written to the registry key HKCU\Software\Policies\System\Wallpaper. This text may be a maximum of 60 characters.

When this policy is *Not Configured* or *Disabled*, this key will not be written.

3) COMBOBOX

Displays a combo box. The **PART** type, **COMBOBOX**, accepts the same options as **EDITTEXT**, as well as the **SUGGESTIONS** option, which begins a list of suggestions to be placed in the drop-down list. **SUGGESTIONS** are separated with spaces and must be enclosed by double quotation marks when a value includes spaces. If a suggestion name includes white space, it must be enclosed in quotes. The list ends with **END SUGGESTIONS**.

For example:

```
SUGGESTIONS
Alaska Alabama Mississippi "New York"
END SUGGESTIONS
```

The valid keywords for **COMBOBOX** are:

-
- KEYNAME
 - VALUENAME
 - DEFAULT
 - SUGGESTIONS
 - REQUIRED
 - MAXLENGTH
 - OEMCONVERT
 - END
 - NOSORT
 - EXPANDABLETEXT
 - CLIENTTEXT
 - END

Displaying a Checkbox on the Property Page of a Policy

CHECKBOX Part Type

Displays a check box. The value is set in the registry with the **REG_DWORD** type.

Default Behavior:

- By default the checkbox is unchecked.
- A check box writes the value 1 to the registry if it is checked, and 0 if it is unchecked.

Syntax

```
PART text CHECKBOX
    VALUENAME value name
END PART
```

text:

This represents the text to be displayed on the right of the check box that you are creating. It can be hard-coded and placed with quotes or you can make the string a variable by putting !! in front of the variable name.

value name

Indicates the registry value to which the selected value will be written. Selecting the option sets the value as a REG_DWORD of 1. Clearing the option removes the registry value. To specify values other than the default values, use the **VALUEON** and **VALUEOFF** statements directly following the corresponding **VALUENAME** statement. These statements are specified as follows:

VALUEON *on value*

VALUEOFF *off value*

When you use these statements, the behavior is modified such that if the administrator selects the option, the value is set to *on value*. If the administrator clears the option, the value is set to *off value*.

CHECKBOX example:

The example in Figure 8 below illustrates the use of **CHECKBOX**.

```
CLASS USER
CATEGORY !!Sample
KEYNAME "Software\Policies"

POLICY !!PartTypesSample
EXPLAIN !!PartTypesSample_Help

; CheckBox (not checked)
PART !!SampleChkBox_NotChked CHECKBOX
VALUENAME "test1"
END PART

END POLICY
END CATEGORY
[strings]
Sample="Desktop Settings"
PartTypesSample="This policy shows the CheckBox Part Type"
PartTypesSample_Help="Sample Part Types"
SampleChkBox_NotChked="Sample CheckBox"
```

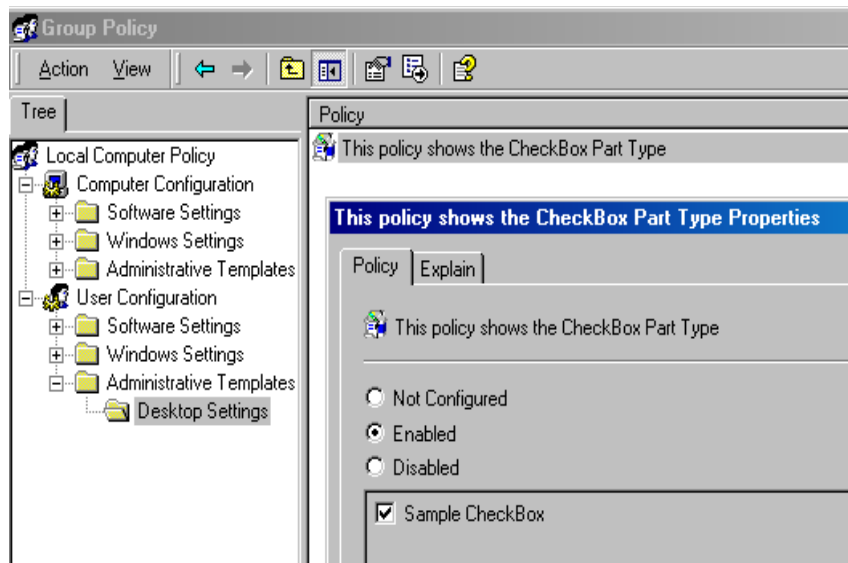


Figure 8. Example of CheckBox.

To override the default behavior

- To have the checkbox checked by default use **DEFCHECKED**.
 - In the sample above, the syntax would be:

```
PART !!SampleChkBox_NotChked CHECKBOX
DEFCHECKED
VALUENAME "test1"
END PART
```
- You can use **VALUEON** and **VALUEOFF**.

- Using the sample in Figure 8 above, the commands below will:
 - Write the string “Enabled” to the registry when the check box is checked.
 - Write a numeric 12 when the check box is unchecked.

```

PART !!SampleChkBox_NotChked CHECKBOX
  VALUENAME "test1"
  VALUEON "Enabled"
  VALUEOFF NUMERIC 12
END PART

```

- To modify more than one registry key you can use an **ACTIONLIST**.

The valid keywords for **CHECKBOX** are:

- **KEYNAME**
- **VALUENAME**
- **VALUEON**
- **VALUEOFF**
- **ACTIONLISTON**
- **ACTIONLISTOFF**
- **DEFCHECKED**
- **CLIENTTEXT**
- **END**

Displaying a Numeric List to the Administrator From Which They Need to Select an Item From the Property Page of a Policy

If you are trying to display a list of numbers, and you would like to have your administrator select one of the predefined numeric values, you should use a spin control. A spin control is implemented using the **PART** type, **NUMERIC**.

1) NUMERIC

Displays an edit field with an optional spinner control (an up-down control) that accepts a numeric value.

Default behavior:

- The value is set in the registry as a **REG_DWORD** type.
- You can optionally have the value written as a **REG_SZ** type by using the **TXTCONVERT** keyword

The **NUMERIC** type accepts the options shown in the following table.

Table 5. Options for NUMERIC.

DEFAULT <i>value</i>	Specifies the initial numeric value for the edit field. If this option is not specified, the field is initially empty.
-----------------------------	--

MAX <i>value</i>	Specifies the maximum value for the number. The default value is 9999.
MIN <i>value</i>	Specifies the minimum value for the number. The default value is 0.
REQUIRED	Specifies that the Group Policy snap-in does not allow a policy containing this PART to be enabled unless a value has been entered for this PART.
SPIN <i>value</i>	Specifies increments to use for the spinner control. The default is SPIN 1 . SPIN 0 removes the spinner control.
TXTCONVERT	Writes values as REG_SZ strings ("1", "2", or "128") rather than as binary values.

Syntax

```

PART text NUMERIC
    VALUENAME value name
    MIN xx
    MAX xx
END PART

```

text:

This represents the text to be displayed on the right of the spin control that you are creating. It can be hard-coded and placed with quotes or you can make the string a variable by putting **!!** in front of the variable name.

value name

Indicates the registry value to which the selected value will be written.

NUMERIC Spin Control Example.

The following example in Figure 9 below illustrates implementing a spin control using the **PART** type, **NUMERIC**.

```

CLASS USER
CATEGORY !!Sample
    KEYNAME "Software\Policies"

    POLICY !!MaxOpenDocs
        EXPLAIN !!MaxOpenDocs_Help
        PART !!Sessions NUMERIC REQUIRED
            VALUENAME "MaxDocs"
                MAX 100
                MIN 1
                SPIN 5
            END PART
        END POLICY
    END CATEGORY

[strings]
!!MaxOpenDocs="The maximum number of connected users"
!!MaxOpenDocs_Help="This policy settings controls the max number of
allowed connections"
!!Sessions="Maximum connections"
!!Sample="Desktop Settings"

```

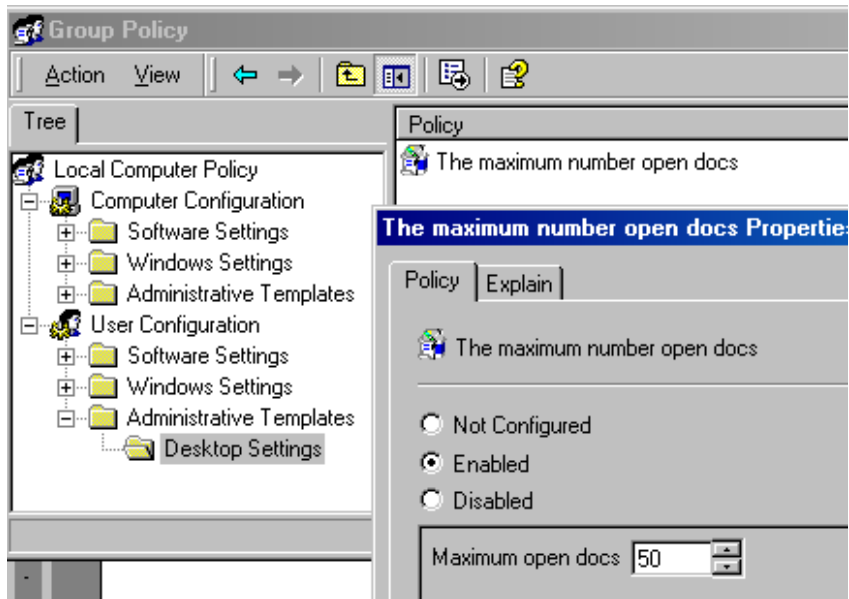


Figure 9. Example of the PART type, NUMERIC.

The valid keywords for **NUMERIC** are:

- **KEYNAME**
- **VALUENAME**
- **MIN**
- **MAX**
- **SPIN**
- **DEFAULT**
- **REQUIRED**
- **TXTCONVERT**
- **END**

- **CLIENTEXT**

Displaying an Alphanumeric List to the Administrator From Which They Need to Select an Item From the Property Page of a Policy

DROPDOWNLIST

You can use this **PART** type to display a combo box with a drop-down style list. You can pre-populate the list of items that are displayed in the list and the corresponding registry value to be written for each item in the list.

DROPDOWNLIST accepts the option shown in the following table.

Table 6. Option for DROPDOWNLIST.

REQUIRED	Specifies that the Group Policy snap-in does not allow a policy containing this PART to be enabled unless a value has been entered for the PART .
-----------------	---

Syntax

```
PART text DROPDOWNLIST
```

```
    ITEMLIST  
        NAME name VALUE value  
        ..  
        NAME name VALUE value  
    END ITEMLIST  
END PART
```

text:

This represents the text to be displayed on the right of the spin control that you are creating. It can be hard-coded and placed with quotes or you can make the string a variable by putting !! in front of the variable name.

name:

This is text that will be displayed in the drop-down list for a particular item.

value:

The value to be written to the specified registry key if this item is selected. Values are assumed to be strings, unless they are preceded by **NUMERIC**. The following example shows both string and numeric values:

```
    VALUE "Some value"  
    VALUE NUMERIC 1
```

DROPDOWNLIST Example:

The following example in Figure 10 below illustrates creating a Drop-down list using the **PART** type, **DROPDOWNLIST**.

```

CLASS USER
CATEGORY !!Sample
KEYNAME "Software\Policies\System"

POLICY !!Autorun
EXPLAIN !!Autorun_Help
PART !!Autorun_Box DROPDOWNLIST REQUIRED
VALUENAME "NoDriveTypeAutoRun"
ITEMLIST
NAME !!Autorun_NoCD VALUE NUMERIC 181 DEFAULT
NAME !!Autorun_None VALUE NUMERIC 255
END ITEMLIST
END PART
END POLICY

END CATEGORY

[strings]
Sample="Desktop Settings"
Autorun_Box="Disable Autoplay on:"
Autorun_NoCD="CD-ROM drives"
Autorun_None="All drives"
Autorun_Help="Disables the Autoplay feature"
Autorun="Disable Autoplay"

```

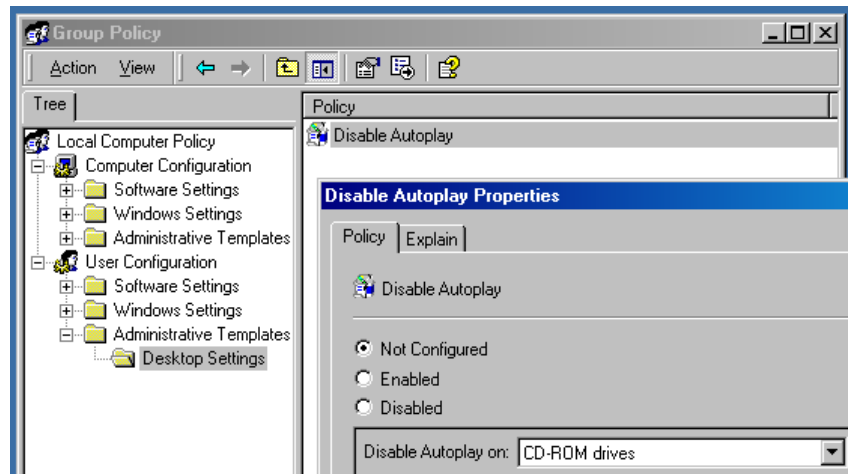


Figure 10. Example of DROPDOWNLIST.

The valid keywords for **DROPDOWNLIST** are:

- **KEYNAME**
- **VALUENAME**
- **REQUIRED**
- **ITEMLIST**
- **END**
- **NOSORT**
- **CLIENTTEXT**

2) LISTBOX

Displays a list box with **Add** and **Remove** buttons. This is the only **PART** type that

can be used to manage multiple values under one key. The **VALUENAME** option cannot be used with the **LISTBOX** part type because there is no single value name associated with this type.

LISTBOX accepts the options shown in the following table.

Table 7. Options for LISTBOX.

ADDITIVE	By default, the content of list boxes overrides any values set in the target registry. This means that a control value is inserted in the policy file that causes existing values to be deleted before the values set in the policy file are merged. If this option is specified, existing values are not deleted, and the values set in the list box is in addition to whatever values exist in the target registry.
EXPLICITVALUE	This option makes the user specify the value data and the value name. The list box shows two columns, one for the name and one for the data. This option cannot be used with the VALUEPREFIX option.
VALUEPREFIX <i>prefix</i>	The prefix you specify is used in determining value names. If a prefix is specified, the prefix and an incremented integer are used, instead of the default value naming scheme described previously. For example, a prefix of " <i>SampleName</i> " generates the value names " <i>SampleName1</i> ", " <i>SampleName2</i> ", and so on. The prefix can be empty (""), which causes the value names to be "1", "2", and so on.

By default, only one column appears in the list box, and for each entry a value is created whose *name* and *value* are the same. For instance, a "name" entry in the list box creates a value called "name" whose data is "name".

The valid keywords for **LISTBOX** are:

- **KEYNAME**
- **VALUEPREFIX**
- **ADDITIVE**
- **NOSORT**
- **EXPLICITVALUE**
- **EXPANDABLETEXT**
- **END**
- **CLIENTEXT**

Appendix B: Additional Keywords

ACTIONLIST

You can use an action list to specify a set of arbitrary registry changes to make in response to a control being set to a particular state.

Syntax

The **ACTIONLIST** syntax is as follows:

```
ACTIONLIST
[KEYNAME key name]
VALUENAME value name
VALUE value
END ACTIONLIST
```

key name

This is an optional path to the registry key. Do not include **HKEY_LOCAL_MACHINE** or **HKEY_CURRENT_USER** in the registry path; the preceding **CLASS** statement defines which of these keys to use. If no key name is specified, the previous key name in the hierarchy is used.

value name

Indicates the registry value to modify. Selecting this option sets the value to a **REG_DWORD** of 1, and clearing the option removes the registry value. If you want to specify values other than the default values, use the **VALUEON** and **VALUEOFF** statements directly following the corresponding **VALUENAME** statement. You specify these statements as follows:

```
VALUEON on value
VALUEOFF off value
```

value

Values are treated as strings unless they are preceded by **NUMERIC**, as in the following examples:

```
VALUE "Some value"
VALUE NUMERIC 1
```

If **VALUE** is followed by **DELETE** (for example, **VALUE DELETE**), the registry entry will be deleted.

The table below contains the two variants for **ACTIONLIST**, that may be used with **POLICY** and **CHECKBOX**.

Table 8. Variants for ACTIONLIST used with POLICY and CHECKBOX.

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

ACTIONLIST Example

The following example illustrates the use of **ACTIONLISTON** and **ACTIONLISTOFF**.

```
POLICY "Deny connections requests"
  EXPLAIN "If enabled, TS will stop accepting connections"
  ACTIONLISTON
    VALUENAME "fDenyTSConnections"    VALUE NUMERIC 1
  END ACTIONLISTON
  ACTIONLISTOFF
    VALUENAME "fDenyTSConnections"    VALUE NUMERIC 0
  END ACTIONLISTOFF
END POLICY
```

EXPANDABLETEXT

Writes a value to registry with data type **REG_EXPAND_SZ**.

For example:

```
PART !!MyVariable    EDITTEXT EXPANDABLETEXT
VALUENAME ValueToBeChanged
END PART
```

REQUIRED

Generates an error if the user does not enter a value when required.

For example:

```
PART !!MyVariable    EDITTEXT REQUIRED
  VALUENAME ValueToBeChanged
END PART
```

MAXLEN

Specifies the maximum length of text.

For example:

```
PART !!MyVariable    EDITTEXT
  VALUENAME ValueToBeChanged
  MAXLEN 4
END PART
```

DEFAULT

Specifies a default value. This can be used for text or numeric data.

For example:

```
PART !!MyVariable      EDITTEXT
  DEFAULT !!MySampleText
  VALUENAME ValueToBeChanged
END PART
```

OR:

```
PART !!MyVariable      NUMERIC
  DEFAULT 5
  VALUENAME ValueToBeChanged
END PART
```

MIN and MAX

Specify the minimum and maximum valid values for a variable.

For example:

```
PART !!MyVariable      NUMERIC
  MIN 100
  MAX 999
  DEFAULT 55
  VALUENAME ValueToBeChanged
END PART
```

KEYNAME

The **KEYNAME** keyword is used within a **CATEGORY** to define which key within the registry is modified as a result of an action here. **KEYNAME** should be followed by the registry path to the key that contains the value that you want to change. Do not specify **HKEY_LOCAL_MACHINE** or **HKEY_CURRENT_USER** because **CLASS** defines that portion of the key. If the **KEYNAME** contains a space, you must enclose the string in quotes.

VALUENAME

Defines the options available within a **POLICY**. First identify the registry value that is to be modified as a result of using the keyword **VALUENAME**. For example, **VALUENAME MyFirstValue**.

The following example illustrates the use of **VALUENAME**. The **Disable Boot / Shutdown / Logon / Logoff status messages** policy prevents the display of system status messages.

```
POLICY !!DisableStatusMessages
  KEYNAME "Software\Microsoft\Windows\CurrentVersion\Policies\System"
  EXPLAIN !!DisableStatusMessages_Help
  VALUENAME "DisableStatusMessages"
END POLICY
```

Unless you specify otherwise, the value is written in the following format when the user checks or clears the option:

- **Checked.** Uses a **REG_DWORD** type with a value of 1.
- **Cleared.** Removes the value.

You can specify options other than these defaults by using [VALUEOFF](#) and [VALUEON](#). If the option is to be selected within the lower pane of the Group Policy snap-in, the **VALUENAME** needs to be within a **PART** scope.

CLIENTTEXT

The **CLIENTTEXT** keyword is used to specify which client-side extension to the Group Policy snap-in needs to process the particular settings on the client computer. By default, the registry extension processes all settings configured under the Administrative Templates node. The **CLIENTTEXT** keyword changes the default behavior and causes the specified extension to process these settings after the registry extension has placed them in the registry.

CLIENTTEXT must be used within either the **POLICY** scope or the **PART** scope and should follow the **VALUENAME** statement.

For example:

```
POLICY !!DQ_Enforce
    EXPLAIN !!DQ_Enforce_Help
        VALUENAME "Enforce"
        CLIENTTEXT {3610eda5-77ef-11d2-8dc5-00c04fa31a66}
    PART !!DQ_EnforceTip1                                TEXT
        END PART
END POLICY
```

The GUID that follows the **CLIENTTEXT** keyword is the GUID of the client-side extension. The client-side extensions are listed in the registry under **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\GPExtensions**.

VALUEON and VALUEOFF

You can use **VALUEON** and **VALUEOFF** to write specific values based on the state of the option. You can enable this functionality by writing the .adm file as described in the following examples:

```
KEYNAME key name
POLICY !!MyPolicy
    VALUENAME ValueToBeChanged
    VALUEON "Turned On" VALUEOFF "Turned Off"
END POLICY
```

OR:

```
KEYNAME key name
POLICY !!MyPolicy
    VALUENAME ValueToBeChanged
    VALUEON 5 VALUEOFF 10
END POLICY
```

Appendix C: Other .adm Topics

Using Simple Policies, and Policies with the VALUEOFF and VALUEON Statements

This section presents two examples that illustrate the difference between using the default policy states and specifying **VALUEON** and **VALUEOFF** statements. There is a significant difference between the two example policies below that you should be aware of.

Example 1

In this example, no explicit **VALUEON** or **VALUEOFF** statements are used. This means that the Administrative Templates will use the default behavior when the user changes the state of this policy.

```
POLICY !!EnableSlowLinkDetect
    EXPLAIN !!EnableSlowLinkDetect_Help
    KEYNAME "Software\Policies\Microsoft\Windows\System"
    VALUENAME "SlowLinkDetectEnabled"
END POLICY
```

The following table lists the default behavior.

Table 9. Example 1 Policy Defaults.

State	Behavior
Policy setting enabled	A DWORD with the value 1 is written to the registry.
Policy setting disabled	The registry value is deleted.
Policy setting not configured	Nothing is changed in the registry.

The important thing to note is the policy-disabled state. The value is not written to the registry with the value of 0; instead it is explicitly deleted. This means a component reading the policy will not find it in the registry, and will fall back to using the default in the code.

Example 2

In this example, the state values are explicitly defined, so when the user changes the policy, the Administrative Templates use these values.

```
POLICY !!EnableSlowLinkDetect
    EXPLAIN !!EnableSlowLinkDetect_Help
    KEYNAME "Software\Policies\Microsoft\Windows\System"
    VALUENAME "SlowLinkDetectEnabled"
    VALUEON NUMERIC 1
    VALUEOFF NUMERIC 0
END POLICY
```

The following table lists the behavior in this example.

Table 10. Example 2 Policy Defaults.

State	Behavior
Policy setting enabled	A DWORD with the value 1 is written to the registry.
Policy setting disabled	A DWORD with the value 0 is written to the registry.
Policy setting not configured	Nothing is changed in the registry.

EXPLAIN

The **EXPLAIN** keyword is used to provide online Help text for a specific Group Policy. In Windows 2000, each policy's **Properties** page includes an **Explain** tab, which provides details about the policy settings.

Each Group Policy that you create should include one **EXPLAIN** keyword, followed by at least one space, and then the **EXPLAIN** string in quotes or a reference to the Help string.

For example:

```
POLICY !!Pol_NoConfigCache
#if VERSION >= 3
EXPLAIN !!Pol_NoConfigCache_Help
#endif
        VALUENAME "NoConfigCache"
        PART !!Lbl_NoConfigCacheHelp1      TEXT
        END PART
END POLICY
.....

[Strings]
Pol_NoConfigCache_Help="Prevents users from changing the\n\nautomatic
synchronization behavior at logoff."
```

In the preceding example, Help is offered for one of the **Offline Files** options. The **EXPLAIN** keyword wrapped in the **#if VERSION** allows this .adm file to be used with the Windows 2000 Group Policy snap-in (which is version 3).

Refer to the section "Writers Role in designing Registry Based Policy" for more details.

Line Breaks

To start text on a new line or to create a line break, use this syntax:

```
\n = Starts a new line
\n\n = Creates a line break
```

#if Version (for Version Comparison)

You can specify that any part of your .adm file be evaluated only in specific versions of the policy editing tools (either the Windows 2000 Group Policy snap-in or the

Windows NT 4.0 System Policy Editor). The latest version for the Windows NT 4.0 System Policy Editor is 2. The first version for the Windows 2000 Group Policy snap-in is 3. To compare versions, use the following syntax:

```
#if Version (operator) x  
#endif
```

The valid **Operators** for the **Version** statement number are listed in the following table.

Table 11. Valid Operators for the Version statement number.

Operator	Signifies
> (GT)	Greater than. For example, $a > b$ means a is greater than b .
< (LT)	Less than. For example, $a < b$ means a is less than b .
== (EQ)	Equal. For example, $a == b$ means a is equal to b .
!= (NE)	Not equal.
>= (GTE)	Greater than or equal to. For example, $a >= b$ means a is greater than or equal to b .
<= (LTE)	Less than or equal to. For example, $a <= b$ means a is less than or equal to b .

Changing the .adm Files Being Used for a GPO

By default, the first time the Group Policy snap-in is started for a specified GPO, it copies the System.adm file from the current computer's %systemroot%\Winnt\Inf directory to the GPO.

Subsequently, only those .adm files specified in the list will be displayed. At each invocation, the Group Policy snap-in also checks the listed .adm files and copies any newer versions from the local computer's %systemroot%\Winnt\Inf directory to the GPO.

Duplicate Category Sections

If the .adm file that you add includes a duplicate **Category** to one that is used in an existing .adm file, the policy settings are merged.

An error may occur under the following conditions. When the existing and the added .adm file contain the same **Category** and both of them have a default **KEYNAME** specified (regardless of whether it is the same name), the following error message is displayed:

"Key name specified more than once. Likely causes are: 1) the KEYNAME tag is defined multiple times in this category, 2) this KEYNAME is already defined in another category with the same name, 3) this KEYNAME is already defined in another category with the same name in a different adm file."

Using Keyboard Shortcuts to Navigate the Administrative Templates Namespace

You can use the following keyboard shortcuts to navigate the Administrative Templates namespace:

- SHIFT+asterisk (*) on the keypad automatically expands the current node and all of its child nodes.
- The plus sign (+) on the keypad expands one level.
- Minus (-) on the keypad collapses one level.
- Double-click on a policy in the results pane to bring up the floating **Properties** page.

When the floating **Properties** page is displayed, move it out from in front of the Group Policy snap-in window. Then click back onto one of the Policies in the results pane. You can now use the cursor keys to navigate up and down the list. Notice that the information in the **Properties** page changes. This method also works for the **Explain** text. You may also use the **Tab** key to move back and forth between tree pane and the results pane, while leaving the **Properties** page open.

Summary

For More Information

For the latest information on Windows 2000 Server, check out our Web site at <http://www.microsoft.com/windows2000> and the Windows 2000/NT Forum at <http://computingcentral.msn.com/topics/windowsnt>.