Version 0.2

How to create custom ADM templates

A first in-sight on how to create a custom ADM template to deploy registry keys with Group Policy.

by Florian Frommherz
Friday, November 2nd, 2007

Table of Contents

Change History

This document is under the process of constant development. I started writing this document as a basic reference on how to create ADM templates. I wanted to use my own words and own examples as – in my opinion – there are many examples and whitepapers out there but only few examples that really show how creating ADM templates and using PARTs really works.

This document may contain minor and/or major mistakes in written language and/or technical specs. Since I'm German and no native speaker/writer, I do my best to provide a readable English-speaking paper here. Any mistakes and misspelled words may be excused.

If you have anything to say about this paper, may it be error-correction, suggestions, wishes or remarks about missing "features" or things that may be included or changed in this paper, feel free to write an email to whitepapers@frickelsoft.net. I highly appreciate your feedback on this, as this is my first public technical paper. If you want to share your thoughts with me, either technically or not, feel free to email me.

Thanks to Norbert and Mark for looking over it!

Also expect constant changes to this paper.

Version	Change Date	Remarks
0.1	16.10.2007	Internal version for my fellow-editors ;-)
0.2	02.11.2007	Fixed some typos, changed an example. - First published version!

Overview

What is this all about?

You all know Group Policy. They're cool when it comes to managing a bunch of computers in your Active Directory environment – if you want to set a default background image, set Internet Explorer's default homepage or something more complex like a customized Office 2003 installation with default settings applied – Group Policy gets you started.

Most of the policies you set are nothing more than registry keys and values. Okay, that was a bit too fast. Some of the settings you can make are managed by Client Side Extensions (CSE). Those are built into the operating system and process settings gathered from the Domain Controllers. A typical CSE that you surely know is the Software Installation. Software Installation is handled by the appmgmt.dll from the Windows operating system (both server and client OSes). It is responsible for processing the data it gets and kicking off Windows Installer with the specific parameters. That is what we call a CSE.

Anyway, in order to be more precise, I need to correct myself: The settings you can use under "User Configuration\Administrative Templates" and "Computer Configuration\Administrative Templates\" are all registry based. All the settings you see there are "created" from those ADM template text files typically located at %windir%\inf. They use the same syntax we will discover just a few pages later. From those ADM template files on, we make setting selections within Group Policy Editor. The GPE creates a registry.pol file that get stored in the SYSVOL-folder on the domain controllers (actually there are two registry.pol files. One for the users, one for the computers). Those registry.pol files will be collected by the (Group Policy) clients and get processed there. All our Group Policy stuff in "Administrative Templates" is based on a bunch of text files containing a weird syntax. From text file templates to real registry settings that get applied by the clients. By the way, the Client Side Extension for this processing is userenv.dll.

Okay, I really should come to the point now. Sometimes you run into an issue that you really need to resolve using a registry key or you just wish to flip that registry key's value from 0 to 1 just to have a certain functionality – and as that is a third party application, there's no setting under "Administrative Templates" that you could use to get it working. Or you just want to set a simple registry key and you don't want to use a registry patch (*.reg) file copied on all computers or executed on all machines, since that registry change could have the need to easily be reverted back to it's original state in a week or two. These are scenarios where you have to create your own Administrative Template and import it into Group Policy Editor. That is what I call the "real magic" about Group Policy. You can do so much with so little administrative effort – if you know how.

What can I do with all this?

Okay, so this is all about creating ADM templates. As I wrote before, those templates are nothing more than text files created in a special syntax that the Group Policy Editor is able to interpret. So it's

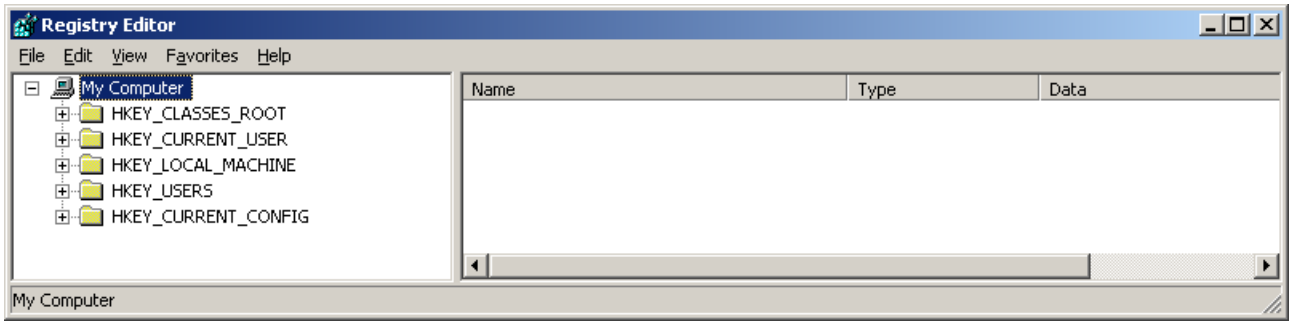
pretty easy to deploy your very own registry settings using Group Policy and Administrative Templates. Almost everything you can handle by firing up the Registry Editor (also known as regedit or regedt32), you can do with Group Policy. By writing “almost” I mean almost, except for two kinds of registry data types that cannot be set by Group Policy administrative templates: those are “REG_BINARY” (binary data) and “REG_MULTI_SZ” (multiline string data). For these two, you’re not lost and alone, you can still export the registry keys into a *.reg-file and deploy them via a startup script – but you cannot set them using an ADM template. Just keep that in mind. For all other registry data types, you’re welcome to make the settings with an ADM template. How it works, comes here.

How ADM templates are structured

What the registry looks like

Everyone knows the Windows Registry. It's a data store similar to a database, where Windows stores information to hardware, software, settings to applications that run on the local machine and user preferences within these applications [1]. Even Windows stores information about settings made in any kind of UI (e.g. the Control Panel) in that place. So as we figured out that ADM templates alter the registry, we should look at the portions of the registry we can reach and what we can do.

The registry is separated into five sections from which you can reach two of them with your templates: "HKEY_LOCAL_MACHINE" and "HKEY_CURRENT_USER".



Picture 1: The Registry Editor

"HKEY_LOCAL_MACHINE" holds machine specific settings. These settings include device driver configuration, environmental configuration and settings that apply to all users that log on to the machine. One can say that HKEY_LOCAL_MACHINE holds the configuration of the computer, hardware drivers and that stuff.

The HKEY_CURRENT_USER portion of the registry saves data for the user currently logged in to the machine. This subkey is actually a "mapping" to the real user's SID under the HKEY_USERS key. When a user logs on, the corresponding node will be mapped in order to be reachable under HKEY_CURRENT_USER. Okay, application can store their user specific stuff here – any options, settings a user can make. We can also store environment variables, network connections and things of that nature. Settings under "HKEY_CURRENT_USER" are settings made for a specific user. Every user on the machine can have a different set of settings.

I think it's somewhat clearer now why all settings are divided into a "Computer Configuration" and a "User Configuration" node. The "User Configuration" is all about the "HKEY_CURRENT_USER" and "Computer Configuration" is about "HKEY_LOCAL_MACHINE". If you had a closer look at the available policies under ..\Administrative Templates\ nodes, you might have recognized that there are a few identical policies on both "User" and "Computer" side. The "computer"-side will have effect on the whole computer, all users that log onto that system will be affected by the changes you make there as the "user"-side only applies to a single user that logs onto some system somewhere in

the forest. See the difference? You can either catch all users that log on to a specific system or you can catch a user to carry the setting with him/her.

The structure of a template

Enough with the talking already, let's see what those templates actually look like. We already discovered that those templates seem to be text files and that they somehow need to differentiate between the "User Configuration" (HKEY_CURRENT_USER) and the "Computer Configuration" (HKEY_LOCAL_MACHINE) portion of the registry. This is handled by the keyword *CLASS*. There is either *CLASS USER* or *CLASS MACHINE*. According to this, the Group Policy Editor figures out where to display the template. But there must be some more information, like the registry key and the value that we'd like to edit.

Let's have a look at the main construction of an ADM template:

```
CLASS xxx
CATEGORY "xxx"
      KEYNAME "xxx"
            POLICY "xxx"
                  VALUENAME xxx
            END POLICY
END CATEGORY
```

Code 1: The basic structure of an ADM template

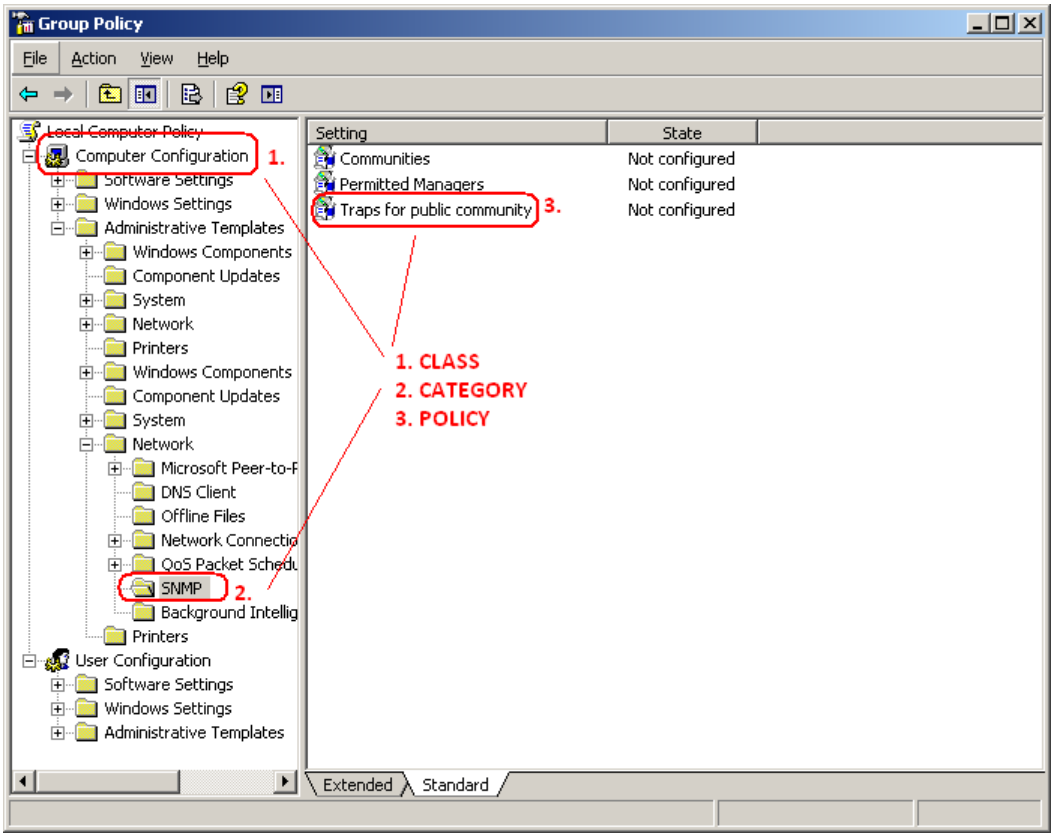
You can see the first line is the „*CLASS*"-line we already talked about. The "xxx" is a place holder for an actual value that we need to specify in order to make the template work. We'll care about that later. Let's first look at the sections we got there so that we may understand the basic setup of a template.

The second line says "*CATEGORY*". This is the place where we specify the category of the policy. For example "Windows Update" or "Offline Files" or something like that. The name we put in here is shown in the Group Policy Editor on the left pane under the corresponding node – "User" or "Computer" – "Configuration".

The "*KEYNAME*" is the real place where the policy will write data into. We provide the key starting from the first node under either "HKEY_LOCAL_MACHINE" or "HKEY_CURRENT_USER" to the last "folder" where our registry value lies in. I'll have a word on the "*KEYNAME*" location that you specify there in a later chapter. A typical keyname would be something like "Software\Microsoft\Office\12.0\Outlook\Preferences" which can be found in HKEY_CURRENT_USER, if you have Outlook 12 installed.

The "*POLICY*" is the name of the policy that is displayed in the Group Policy Editor. We'll give our policies cool names so that we can see on the first sight, what the policy actually does. This name is the name that users will see on the right pane of the Group Policy Editor.

Last but not least, we've got "VALUENAME" there, which is the value that we want to change in the registry. We assign values new "*CLASS*", "*KEYNAME*" and "*VALUENAME*" uniquely identify a registry value in its path and location.



Picture 2: The Local Group Policy Editor showing some Computer Configuration settings.

Having this knowledge now, we could write ourselves a small ADM template that would have a very basic functionality and change a registry key. Keep in mind that the structure is the minimum an administrative template needs to have. Our template would now only give the user the ability to select "Enabled" or "Disabled" (as well as "Not Configured"). If we want our template to provide users the ability to select values from a textbox or to choose from a DropDownList, we'll have to use a more complex structure and elements in so-called "parts". What and how, we'll see in a later chapter.

Creating the very first ADM template and importing it

Looking at a live example

After we had a look at the basic structure of an ADM template, we now want to see what a "real-world-template" looks like. I'll show you a valid ADM template that you could import right-away to your Group Policy Editor for usage. We then see what parts of the template are still unfamiliar to us and have a look at the purpose of those new keywords and elements.

```
;Configure the Windows Search. Will we use that Search Assistant?
CLASS USER

CATEGORY "System"
CATEGORY "Windows Search"

          KEYNAME "Software\Microsoft\Windows\CurrentVersion\Explorer\CabinetState"
          POLICY !!searchAssistant
          EXPLAIN !!ExplainWords
                    PART "Show Search Assistant?" DROPDOWNLIST
                    VALUENAME "Use Search Asst"
                              ITEMLIST
                              NAME "Disable Search Assistant" VALUE "no"
                              NAME "Enable Search Assistant" VALUE "yes"
                              END ITEMLIST
                    END PART
          END POLICY
END CATEGORY
END CATEGORY

[strings]
searchAssistant="Show Search Assistant when searching in Windows"
ExplainWords="This policy enables/disables the Windows Search Assistent"
```

Code 2: A more complex ADM template for Windows search

Phew, that looks a little more advanced, doesn't it? Okay, let's see what we got there. We have our "CATEGORY" section (twice!) which says that the policy goes in the left Group Policy Editor pane under "Administrative Templates\System\Windows Search". We have our "POLICY" keyword with some sort of cryptic value behind it, starting with a double exclamation mark. A "KEYNAME" is also there, which tells us that the setting is going to alter a value in the "User Shell Folders" sub node of "HKEY_CURRENT_USER". That's because we have the "CLASS USER" keyword. As a "VALUENAME" we've got "Use Search Asst" there. Everything we need for a basic ADM template is there – but there's much other stuff in this one. Let's go line by line and see what we haven't discovered yet.

The first line starts with a semi-colon. This indicates a comment. You can comment your ADM templates so that other may better understand what's going on with it. Everything that's written on the right side of a semi-colon (;) in an ADM template is considered a comment and will not be viewed when applying the template. Some people create comment-headers at the beginning of an ADM template to manifest the creation date and the author of an ADM template. That makes sense if you are a big company and need to know when the template was last review and by whom.

The next lines provide nothing special, except for the double "*CATEGORY*" implementation here. I did not format these categories in Code 2 right. I should have written those in two different columns to make it a little clearer. Just like this:

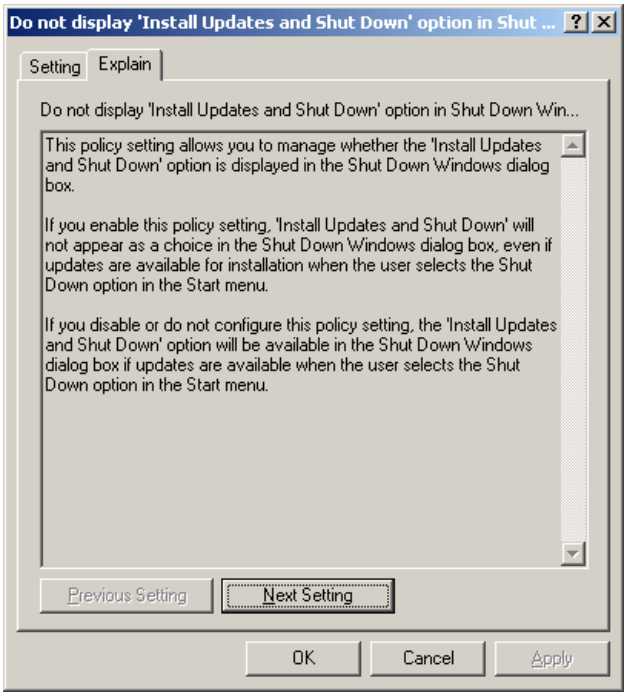
```
CATEGORY "System"
        CATEGORY "Windows Search"
        …
        END CATEGORY
END CATEGORY
```

Code 3: Formatting ADM templates right increases readability.

…instead of writing the two categories in the same column. You can put "CATEGORY" sections into other "CATEGORY" sections. You can create a folder structure in the left pane of the Group Policy Editor - like the one you can see in Picture 2. You can see we've got a category called "Network" and then various subcategories like "DNS Client" for example. So don't be afraid of the categories – make use of them and structure your templates!

The line starting with "*POLICY*" is pretty interesting for us. The "*POLICY*" keyword is followed by a strange looking string "*!!searchAssistant*". This is a variable or a place-holder – whatever you will call it. It is a placeholder for a string that is specified at the end of the ADM template under the so-called [strings]-section. A string always starts with two exclamation marks (!!). Strings are there for making the template look better and it stores all texts at a central place at the end of the template. Variables can be used with our favorite keywords "*CATEGORY*", "*POLICY*", "*PART*" and "*EXPLAIN*" which we want to get to know right now. Doing this makes localization and translations much easier as people don't have to crawl through the whole template rather than looking into the [strings] section and translate everything given there.

The keyword "*EXPLAIN*" is somewhat self-explaining (har!). It adds some help text to the policy in the Group Policy Editor and can be used just to make things clear. Sometimes a policy name or a registry key doesn't exactly explain what the setting is doing or what effects the setting will have on target systems. It is therefore great to have a text that briefly explains what will happen to the system if people "enable" or "disable" the policy. Use the "*EXPLAIN*" keyword for each and every policy you create helps people understand, what the policy is actually out to do. If you have a look at the ADM templates shipped with Windows, you'll see that they're often really detailed. They explain what happens if you set policyX to "Enabled", what changes if you "disable" it and what the default state of "Not Configured" is. "*EXPLAIN*" is a powerful friend – use it!



Picture 3: Looking at an explain text of a Group Policy.

Also new to us are the two keywords "*DROPDOWNLIST*" and "*ITEMLIST*". The "DROPDOWNLIST" keyword advises the Group Policy Editor to provide the user a list where he/she can select an option from a predefined list. This "*DROPDOWNLIST*" is a so-called "part" (which is why it is enclosed in "PART" and "END PART" tags) we will talk about a little later.

The keyword "*ITEMLIST*" allows us to define the items that are displayed within that *DROPDOWNLIST*. We'll se what this is all about, later.

Optional statements

There are a few more statements and keywords that you might want to use. Looking at our first example, it seems like we can only specify a VALUENAME within the registry – no values, nothing. If you browse the registry from time to time, you surely noticed that there are many values, VALUESNAMEs can apply. Our first ADM template doesn't seem to do any of these things right now, let's see how we can get a little further:

There's a "*VALUE*" keyword which can be used for setting the value that is written into the registry when the policy is set to "Enabled" or "Disabled". If we leave the "*VALUE*" keyword out, like we do in our example, the policy will create a "DWORD" with the value 1 when checking "Enabled" and delete the whole value including the value name when checking "Disabled" at the policy. To alter the default behavior and to be able to set a custom value when checking "Enabled", we need to provide the "VALUE" keyword, followed by the value we want to set. Like in this example:

```
…
KEYNAME "Software\Examples\Playground"
            VALUENAME DarlingsNickname
```

```
                    VALUE "Purzel"
…
```

Code 4: An example for the "VALUE" keyword.

In this example, if my darling had the nickname "Purzel", I'd open the policy and click "Enabled". The policy would then set "DarlingsNickname"'s value to "Purzel". If I went to set the policy to "Disabled", the value as well as the valuename would get deleted. If we provide the *VALUE* like this, it gets automatically written as a REG_SZ value. Good thing, but if we wanted to have a number rather than a string be set in the registry, we'd need to use the "*NUMERIC*" keyword right after *VALUE*. See:

```
…
KEYNAME "Software\Examples\Playground"
            VALUENAME DistanceQueryMode
                  VALUE NUMERIC 2
…
```

Code 5: An example for the "VALUE" keyword – now using "NUMERIC" as well.

Now, we'll have a DWORD value (remember: a DWORD is just a number) set to 2 if we "Enable" the policy. If we "Disable" it, it will still be deleted. That's good for many situations. But what if we need to have the value be set to a certain value thing rather than deleted when "Disable" the policy? This is where "*VALUEON*" and "*VALUEOFF*" keywords come into play.

Using VALUEON and VALUEOFF, you can choose which values the policy write into the registry when clicking "Enabled" or "Disabled". By default, the values you provide will be interpreted as "REG_SZ" – if you want numbers, the "NUMERIC" keyword still applies for you. Here's another example:

```
…
KEYNAME "Software\Examples\Spielwiese"
            VALUENAME ScrewdriverDirection
                  VALUEON "Left"
                  VALUEOFF "Right"
…
```

Code 6: You can use "VALUEON" and "VALUEOFF" if you need non-default values.

For VALUEON and VALUEOFF, you can also use another keyword, called DELETE. By specifying "VALUEON DELETE" or "VALUEOFF DELETE", you can advise the system to simply delete the value. "DELETE" will wipe the valuename as well as the value itself out of the registry.

Okay, we're now able to set a certain valuename to a specific value, both string and integer (number). What we now want to come up with is a little more advanced: what if we need to change multiple valuenames with a single policy? You set a policy to "Enabled" and have multiple registry keys that you need to change? This is what "*ACTIONLIST*" is good for. As the name implies, the ACTIONLIST tags surround a list of valuenames and values that need to be altered when enabling or disabling a policy. The exact keywords are "*ACTIONLISTON*" and "*ACTIONLISTOFF*". If you look at the example code 6, it's pretty clear:

```
…
KEYNAME "Software\Examples\Spielwiese"
       VALUENAME "UseColoredBackground"
              VALUEON NUMERIC 1
       ACTIONLISTON
```

```
                VALUENAME "color" VALUE "255, 255, 255"
                VALUENAME "image" VALUE "myImage.bmp"
                VALUENAME "style" VALUE NUMERIC 2
        END ACTIONLISTON
…
```

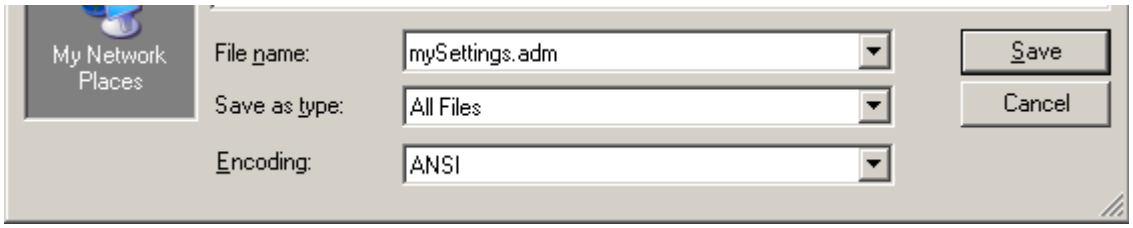
Code 7: Using the ACTIONLISTON keyword

If we set the policy to "Enabled", the "Examples\Spielwiese" subnode will be created in the registry if it hasn't already been there. A DWORD called "UseColoredBackground" with value 1 will be created as well as all other valuenames you can see within the "ACTIONLISTON" (and "END ACTIONLISTON") tags. So if we enable that policy, four valuenames get created and set to the corresponding values. Analog to that, if we disable to policy, those four valuenames get wiped out the registry. If we wanted to have another set of valuenames set when we disable the policy, we could – of course – use "ACTIONLISTOFF" and "END ACTIONLISTOFF" to define our settings.

Saving and importing our ADM template

Okay, now that we've seen the basic functionality of an ADM template and discovered the mysterious basic keywords that come with it, we're able to save and import our ADM template.

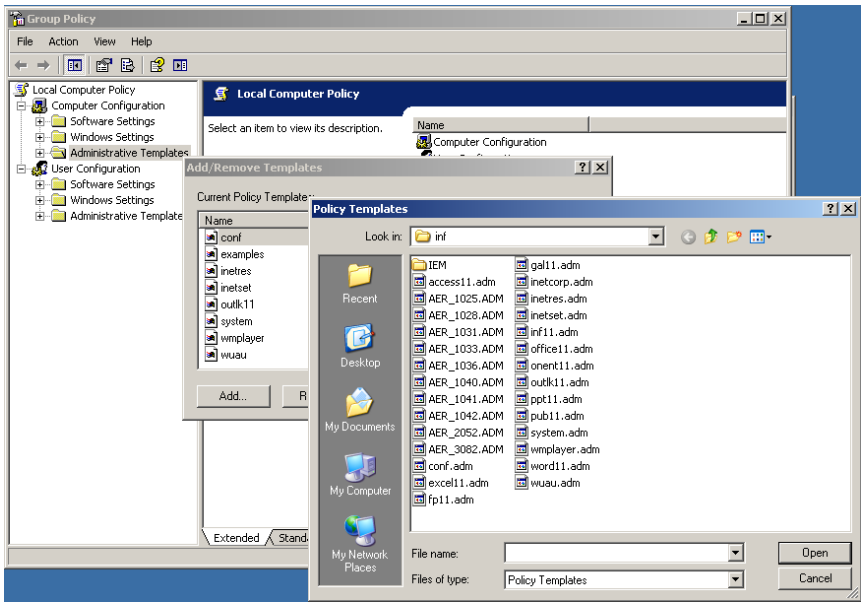
Since ADM templates are text files, you can simply start notepad.exe from the command line and hack the keywords in there. It doesn't need much more although there are a few third party ADM template editors out there (some of them aren't free, be careful!). I typically use notepad and hack my stuff in there, but you can have a look at conTEXT [3], for example. It's a free text editor with highlighting extensions available [4]. If you do the same and you're ready to save the template, put it into a location where you will find it again. When saving it, be sure to pick "All files" in order to be able to save it as an *.adm file (and not an .adm.txt file). The location where the Group Policy Editor looks at first when importing an ADM template file is the %windir%\inf folder. This is where all Microsoft shipped ADM templates lie in.



Picture 4: How to save the ADM template the right way.

After you saved the file as *.adm, open up the Group Policy Editor (it doesn't matter if you do it locally or with "Active Directory Users and Computers or using GPMC). When you right-click the "Computer Configuration" or "User Configuration" folder you should see the option "Add/Remove administrative templates" – there we go. After clicking "Add…" you can browse for your custom ADM template and choose "Open". You can then close the "Add/Remove Templates" window. If your template is well formatted and the Group Policy Editor was able to "parse" it, there won't be any

message. You can now browse through the nodes and find your settings. If you however did a mistake, did not well-format your template or chose the wrong key words, you will be presented with an error message telling you where and what went wrong. In this case, the template will not be imported unless you correct the error and try it again.



Picture 5: Import your ADM template

When you successfully added your ADM template (that is, if you're not getting error message right after your import) and you browse through the "Administrative Templates" nodes and you can see the category that you specified, but cannot see the settings inside, you ran into a common problem. I refer to that problem in the "Sometimes, things go wrong" section.

Using #if version

With nearly every Windows release, Microsoft released a new version of the Group Policy Editor. Some features of newer versions of the Group Policy Editor, for example the SUPPORTED-keyword which we'll discover in the next section, cannot be viewed and interpreted with earlier versions of the Group Policy Editor.

With #if version, you can decide which portions of your ADM template can be viewed and interpreted by a certain version of Group Policy Editor and which cannot. The syntax is:

```
…
#if version > 3
[statements]
#endif
…
```

Code 8: if version in action

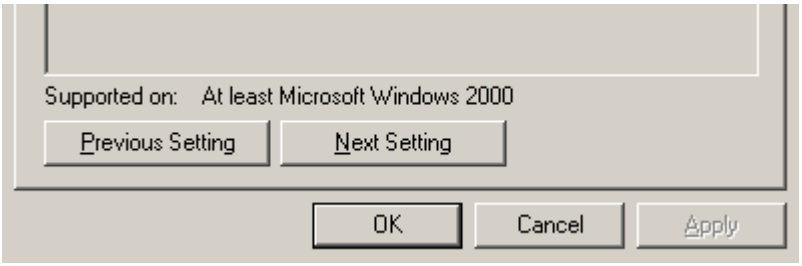
In this example, if the client's GPE version is greater then 3, the statements between #if version and #endif will be executed. If the version equals 3 or is less than that, the statements will be ignored. You can use all known operators like <, >, >=, <=, == and !=

The versions of GPE are as follows [2]:

Version 1.0	Windows 95
Version 2.0	Windows NT 3.x and Windows NT 4.x
Version 3.0	Windows 2000
Version 4.0	Windows XP and Windows Server 2003
Version 5.0	Windows XP SP2.

Using the SUPPORTED-keyword

You surely already noticed the "Supported on: .." section at the bottom of Microsoft standard Group Policy settings:



Picture 6: The SUPPORTED-keyword

If settings you want to provide an ADM template for require a specific version of Windows or another specific software installed, you can create your own "Supported on:…" message. Our keyword is "SUPPORTED" and is used like this:

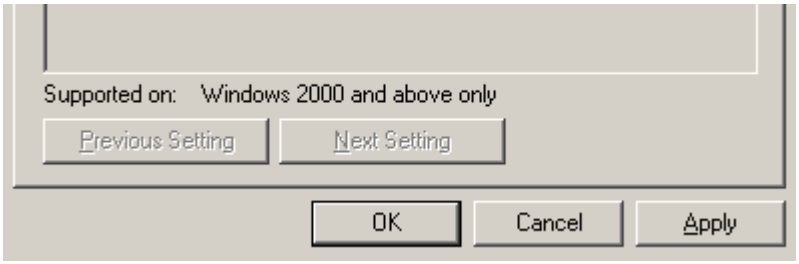
```
CLASS USER

CATEGORY !!categoryName
 POLICY !!policyName
  KEYNAME "Software\Spielwiese\FSFT\Testing"
  SUPPORTED !!mySupported
  VALUENAME "myTestValue"
       VALUEON NUMERIC 2
       VALUEOFF NUMERIC 0
END POLICY
END CATEGORY

[strings]
categoryName="my custom ADM templates"
policyName="A simple SUPPORTED check policy"
mySupported="Windows 2000 and above only"
```

Code 9: Using SUPPORTED keyword

Okay, we have a user policy that will actually write 2 into some registry key when enabled and 0 if disabled – but that's not what's interesting to us. You see the "SUPPORTED !!mySupported" keyword



on line 6. Under [strings] we have our supported string that will be displayed this way:

Picture 7: Our custom SUPPORTED section in action

A pretty easy thing but essential if other users shall use your custom ADM template and (parts of) your settings trigger a special version of an application.

Using parts to let the policy look cool

Why parts make ADM templates much more fun

When we look at the ADM template we just created, we are able to do the following things: we can choose between “enabled” and “disabled” states in which we can specify a predefined value for a valuename in a certain data type. We can also alter multiple valuenames with a single click on “enabled” or “disabled” when we use the actionlists.

You surely noticed the word “predefined” in the sentences above. If you think of the templates we were to talk about until now, you come to see that we always had predefined settings for the states “enabled” and “disabled”. We always provided the values in a static manner in our ADM template. What do those ADM templates look like that Microsoft shipped with Windows? Those cool settings we can use to put in custom data or text or use some checkboxes or whatever. All the cool stuff that makes an admin’s life worth living, where is it?

Help is near, my friend! The next few sections are all about “PARTs”. Right, parts are these “controls” that you can add to your template to give administrators out there a little more control over things. We’ll go through every part that’s available – one by one. In the end we’ll combine them and see what magical things we can do.

The TEXT part

The TEXT part is the easiest part we can use. Since it’s very easy it’s also not much useful. You can display a text message or an explanation in the policy – but you cannot alter any registry values with it. It’s just a “visual” feature.

Let’s have an example. We want to assign the “.TIF” file extension a new application:

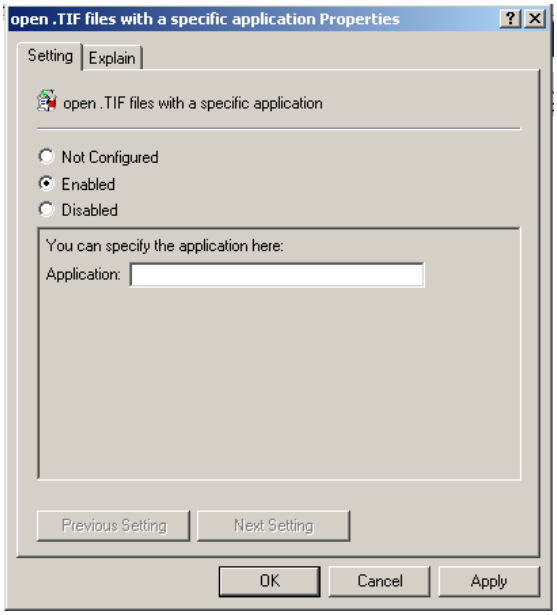
```
CLASS USER

CATEGORY !!categoryName
       POLICY !!policyName
              KEYNAME "Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\.tif"
                     PART "You can specify the application here:" TEXT
                     END PART
                     PART "Application:" EDITTEXT REQUIRED
                     VALUENAME "Application"
                     END PART
       END POLICY
END CATEGORY

[strings]
categoryName="my own Windows Explorer settings"
policyName="open .TIF files with a specific application"
```

Code 10: Use the “TEXT”-part to write some words into your template

The interesting lines in this template are printed bold. Just have a look at those. The first PART – END PART tags contain the "*TEXT*" part. They have the "You can specify the application here:" text which will be displayed when you open the setting:



Picture 8: The TEXT-part in action

As said before, you can use the "*TEXT*"-part to provide some words of wisdom, warnings and help within your setting.

The EDITTEXT part

The *EDITTEXT* part is much more fun. If you had a closer look at the Code 7-ADM fragment, you recognized a second PART, an *EDITTEXT* part. *EDITTEXT* displays a textbox in your setting which lets and administrator enter some characters.

```
CLASS USER
…
        PART "Application:" EDITTEXT REQUIRED
        VALUENAME "Application"
        END PART
…
```

Code 11: A closer look at an EDITTEXT fragment

So when looking exactly at our fragment here, we can see its pretty straight-forward. We have our EDITTEXT part with a VALUENAME. This leads to the fact that whatever we enter into the textbox, it will be written into the registry key "Application" as REG_SZ – as a string. You can write a max of 255 characters into that text box.

This and a few more things can be handled with optional settings. Just like the "*REQUIRED*" keyword you see after "*EDITTEXT*". "*REQUIRED*" is there to make sure you really enter a string into the textbox. If we'd "Enable" our policy and leave the textbox blank without having "*REQUIRED*" specified in our template, the setting would be fine and be written into the registry. So if you need to make sure a value is specified in the textbox, add "*REQUIRED*".

You can also set a "*DEFAULT*" value for the *EDITTEXT* textbox when people open setting. Just specify "*DEFAULT "my text""* as keywords into the PART tags and it works.

If you need to limit the possible characters from 255 (default) to a smaller number, let's say 80, you can use the *MAXLEN* keyword. With *MAXLEN 80* you restrict the use of more than 80 characters.

If you wish to use environment variables in your setting and the *EDITTEXT*, you can use the keyword "*EXPANDABLETEXT*" – this forces the setting to save the text from the textbox to be stored in the registry as REG_EXPAND_SZ.

So after all the optional keywords, our code would now look like this:

```
CLASS USER
…
        PART "Application:" EDITTEXT REQUIRED DEFAULT "notepad.exe" MAXLEN 30 EXPANDABLETEXT
        VALUENAME "Application"
        END PART
…
```

Code 12: Our EDITTEXT fragment customized

The DROPDOWNLIST part

Let's imagine we have our template from Code 7 which lets us choose an application to open any file with the .tif extension by default. Let's further imagine we now want to set a default application to open any files with Administrative Templates extension, .adm – and let's further imagine we want to let any user who uses our setting choose between three default application we specify. How about that? Well, we could use the DROPDOWNLIST part to display the user the possibilities he might pick:

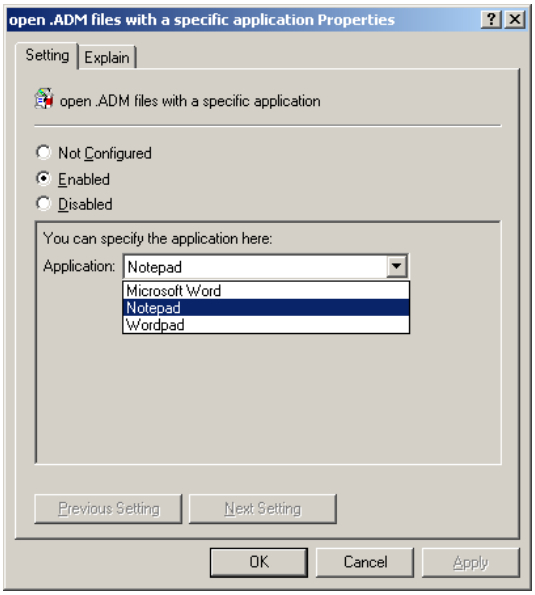
```
CLASS USER

CATEGORY !!categoryName
 POLICY !!policyName
  KEYNAME "Software\Microsoft\Windows\CurrentVersion\Explorer\FileExts\.adm"
  VALUENAME "Application"
  PART "You can specify the application here:" TEXT
  END PART
  PART "Application:" DROPDOWNLIST
       ITEMLIST
              NAME "Notepad" VALUE "notepad.exe"
              NAME "Wordpad" VALUE "wordpad.exe"
              NAME "Microsoft Word" VALUE "wordpad.exe"
       END ITEMLIST
  END PART
 END POLICY
END CATEGORY
```

```
[strings]
categoryName="my own Windows Explorer settings"
policyName="open .ADM files with a specific application"
```

Code 13: Using the DROPDOWNLIST-part for an ADM-Template to control the application to open *.adm files

Without further comment, I show you what this template looks like:



Picture 9: The TEXT-part in action

We did specify our *DROPDOWNLIST*-part as you can see. After that, we set the *VALUENAME* which is still "Application" – we want to set that to a new value, just like before. Now, with the *DROPDOWNLIST*, we're able to present the user with a handful of application from which he or she can pick one. We do that with the *ITEMLIST*-keyword. "*ITEMLIST*" and "*END ITEMLIST*" surround options we want to display to the user. The "*NAME*" keyword is the displayed name the user gets to see to make his/her choice. The "*VALUE*" is still the value that will be written into the registry.

Again, we have some optional keywords we can use in order to make our setting more fun. Just like we used the "*REQUIRED*" keyword in *EDITTEXT*, we can use it here, too. By specifying "REQUIRED", users need to make a decision and choose one of the options we gave them.

If you had a really close look at the template and the example picture above, you can see that in the picture, the options are sorted alphabetically. Group Policy Editor does that by default. If we don't want that since we want to keep the order we wrote our options into the ADM template, we can use the keyword "*NOSORT*". *NOSORT* directs the Group Policy Editor to just leave the sorting alone and use the sorting we made within the ADM template.

The NUMERIC part

Okay, enough with the text and strings already. It's time to dive into the world of numbers - but not too far though. For the NUMERIC part, I'd like to have a look at some infamous settings, Microsoft shipped with Windows. The setting I'm talking about has its own KB article here:

http://support.microsoft.com/kb/290324. It's a policy setting to define the max profile size for users. The ADM fragment from the KB article and in the system.adm template on all of your machines looks like this:

```
…
        PART !!ProfileSize NUMERIC REQUIRED SPIN 100
        VALUENAME "MaxProfileSize" DEFAULT 30000
        MAX 30000
        MIN 300
        END PART
…
```

Code 14: The infamous max profile policy setting using the NUMERIC part.

The !!ProfileSize-variable doesn't bug us at the moment. If you open the corresponding policy in your Group Policy Editor at User Configuration\Administrative Templates\System\Logon/Logoff\ - "Limit profile size", you can see how this setting actually works.

If you enable it, you can see that 30000 is set in the part. That part seems to be our *NUMERIC*-part. If we use the arrows up and down at the right side of the part, we can decrease or increase the number in our part – every time we click it, it changes by exactly 100. We can also enter a custom number by clicking into the numeric-part and typing in a value. We cannot choose a number higher than 30000, nor can we use a number less 300. Not look at the template again.

We pretty much have all our keywords explained ourselves. The *NUMERIC*-keyword is our part itself and we know *REQUIRED* from other parts as well. The *SPIN*-keyword helps us to specify how far the number shall change every time we click the arrow-buttons. So we can here specify the steps to take on every click. The DEFAULT-keyword can be used to set an initial value when enabling the policy. We know this from other parts. The MAX and MIN-keywords are optional. You can set the min and max value for the spin. It will stop at these values and not allow the closing of the setting, if a value outside of these min/max ranges is specified (by hand).

As you probably have guessed, the values chosen here will be written as DWORDs (Integers). There's an optional keyword called "*TXTCONVERT*" which takes care of that. If you want the value to be written as a REG_SZ string, use *TXTCONVERT*.

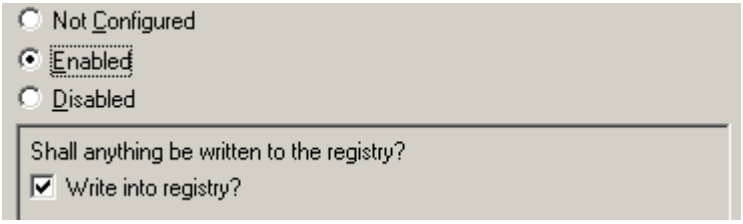
The CHECKBOX part

Well all know checkboxes. They're quite cool as they can only have to states: checked and unchecked. That should be easy to understand. Let's see what the basic syntax looks like:

```
…
  PART "Shall anything be written to the registry?" TEXT
  END PART
  PART "Write into registry?" CHECKBOX
         VALUENAME "myCustomEntry"
  END PART
…
```

Code 15: A very basic CHECKBOX example.

Okay, we just look at the very basic PART here; forget the upper PART with our "TEXT"-part. You can see we've got our CHECKBOX part ready to go with no options behind it. In the part, we've got our VALUENAME, "myCustomEntry". If we "enable" that policy now, we can check the checkbox or leave it unchecked.



Picture 10: Our very basic CHECKBOX example.

If we check the checkbox and click "apply", by default, 1 as a number (a DWORD) is written into the registry. If the box isn't checked, 0 is written into the registry. To overwrite the default behavior, we can use our friends *VALUEON* and *VALUEOFF*:

```
…

  PART "Shall anything be written to the registry?" TEXT
  END PART
  PART "Write into registry?" CHECKBOX
         VALUENAME "myCustomEntry"
         VALUEON NUMERIC 2
         VALUEOFF NUMERIC 1
  END PART
…
```

Code 16: Our very basic CHECKBOX example with our VALUEON/VALUEOFF friends.

In Code 13, when checked, we'll write a 2 (as a number) into the myCustomEntry key, a 1 if it's unchecked. To have our template write strings into the registry, we simply use our "-signs:

```
…

  PART "Shall anything be written to the registry?" TEXT
  END PART
  PART "Write into registry?" CHECKBOX
         VALUENAME "myCustomEntry"
         VALUEON "enabled"
         VALUEOFF "disabled"
  END PART
…
```

Code 17: Our very basic CHECKBOX example with our string friends.

We can even use more optional keywords. Our friends "*ACTIONLISTON*" and "*ACTIONLISTOFF*" can also be used to define multiple actions when the box is checked or unchecked. I like that.

We can also define whether the box shall be checked by default or not. We use the "*DEFCHECKED*"-keyword for that:

```
…
```

```
  PART "Shall anything be written to the registry?" TEXT
  END PART
  PART "Write into registry?" CHECKBOX DEFCHECKED
         VALUENAME "myCustomEntry"
         VALUEON "enabled"
         VALUEOFF "disabled"
  END PART
…
```

Code 18: Our very basic CHECKBOX example with DEFCHECKED.

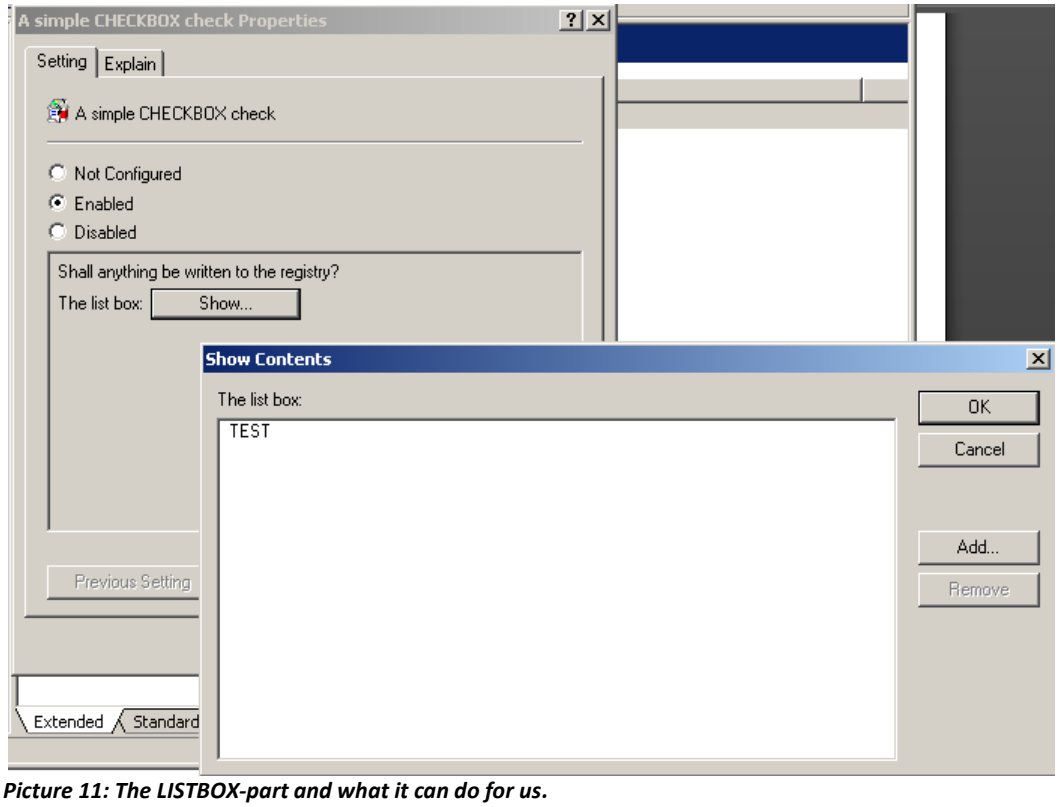
The LISTBOX part

The LISTBOX-part is one of the parts that has the simplest syntax:

```
…
  PART "The list box:" LISTBOX
  END PART
…
```

Code 19: See how easy the LISTBOX part is to implement.

And this is the result:



Picture 11: The LISTBOX-part and what it can do for us.

You can see, where we would expect our LISTBOX-part, there's a "Show" button. When we click it, we can see a "Show Contents" window. This is a list of VALUENAMES that are in our registry under the KEYNAME we specified in our ADM template. You surely have noticed that we didn't specify a *VALUENAME/VALUE* in the ADM-fragment above. The *LISTBOX*-part doesn't support those. It is made to display all valuesnames it finds under the given Keyname. It is the only part with which you can

both create and delete valuenames by using the “Add…” and “Remove” buttons at the right side. When clicking “Add…” and specifying a valuename, it will be put into the registry. Be sure to know that all you see in the popup window will be put into the registry – it will overwrite all valuenames and settings that have been there before. “Former” valuenames will be replaced by the ones you specify.

To change that, we can use the “*ADDITIVE*”-keyword which directs the system to append the valuenames and values we specify here, instead of replacing the old ones with the new ones:

```
…
  PART "The list box:" LISTBOX ADDITIVE
  END PART
…
```

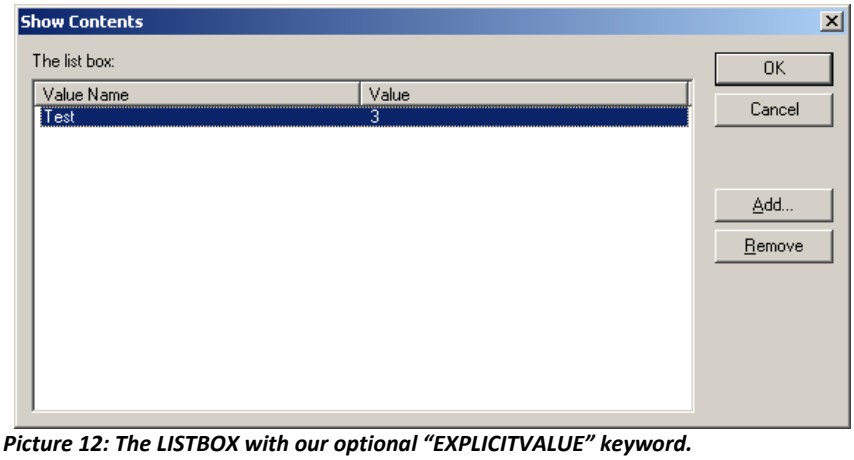
Code 20: Add a prefix to every newly added valuename.

We can use the “*EXPLICITVALUE*” keyword as an optional keyword to have the popup window display a second column that allows us to view and set the values to the valuenames we see:

```
…
  PART "The list box:" LISTBOX EXPLICITVALUE
  END PART
…
```

Code 21: See how easy the LISTBOX part is to implement.

Using *EXPLICITVALUE*, it looks like that:



Picture 12: The LISTBOX with our optional “EXPLICITVALUE” keyword.

Another useful optional keyword is “*VALUEPREFIX*” – we can specify a prefix for every value made in the *LISTBOX*. By simply doing this:

```
…
  PART "The list box:" LISTBOX VALUEPREFIX "NewVal"
  END PART
…
```

Code 22: Add a prefix to every newly added valuename.

every new valuename that is created in the *LISTBOX* will have the prefix “NewVal”, so adding “Test” will result in “NewValTest”. Note that you cannot use the *VALUEPREFIX* and *EXPLICITVALUE* keywords together.

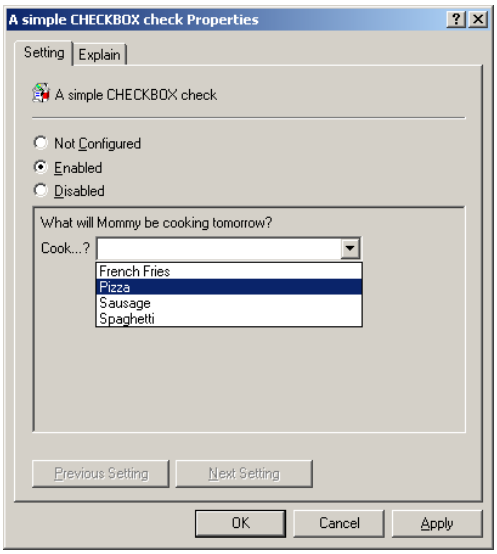
The COMBOBOX part

COMBOBOX is the last one we’re looking at. This will be an easy one for us as well, since we know its parents pretty well. *COMBOBOX* is a combination of *EDITTEXT* and *DROPDOWNLIST*. It’s basically a DROPDOWNLIST into which users can put custom values by clicking into it and typing any value in there. Just like with a DROPDOWNLIST, we’re able to “preset” some options for the user to choose from, but they’re called suggestions now. “SUGGESTIONS” is the keyword for this as well. Here’s a basic example:

```
…
  PART "What will Mommy be cooking tomorrow?" TEXT
  END PART
  PART "Cook...?" COMBOBOX
  VALUENAME "MommyShallCook"
        SUGGESTIONS
               "Spaghetti" "Pizza" "Sausage" "French Fries"
        END SUGGESTIONS
  END PART
…
```

Code 23: A basic COMBOBOX example that shows its functionality.

which looks like:



Picture 13: The Our COMBOBOX example

As we can see, our *COMBOBOX* enables us to specify a custom value that will be written into the registry if we don’t want our suggestions. The predefined suggestions are between the *SUGGESTIONS* and *END SUGGESTIONS* tags each separated by a space. We could also use !!variables in order to

make our ADM-template localizable. Just keep that in mind - !!variables are cool, use them as often as you can!

Like with the other Parts, we have some optional keywords we can use: Just like the DROPDOWNLIST, we have the "*NOSORT*"-keyword which will leave our sorting from the ADM template alone. If we don't specify it, our suggestions will be sorted alphabetically.

We can also use the "*MAXLEN*" keyword which limits the characters of the users input. The default value is 255 characters.

The "DEFAULT" value can specify the default value that is written in the COMBOBOX-field when users open the setting.

It also handles the "REQUIRED" and "EXPADABLETEXT" keywords we already know from EDITTEXT.

Sometimes, things go wrong. What then?

The [strings] section got truncated!

This error is a classic. Former versions of the Group Policy Editor have problems with interpreting some string types that are larger than 255 characters. You can read more about that problem in the KB 842933: http://support.microsoft.com/kb/842933.

I imported the template and can see the category. But where the heck is the policy?

You probably created a Preference, not a Policy. The Group Policy Editor only shows Policies by default. To quick-resolve that issue, right-click your category-folder at the left side of your Group Policy Editor, choose "View"-"Filtering" and uncheck both checkboxes at the bottom the window that say "Only show…"

See the following sites about this problem and why this is:

Darren "GPOGuy" Mar-Elia's Website: http://www.gpoguy.com/FAQs/tattoo.htm
Florian Frommherz' Blog: http://www.frickelsoft.net/blog/?p=8

I imported the template and tried it out. But nothing happens. Why?

That can have multiple reasons. When applying the policy to an OU or a local computer, you need to inspect, if the policy gets applied. Try to run rsop.msc on a domain client, to see if the machine is really in the scope of the Group Policy you made the changes to.

If the policy is applied, have a look at the registry and see if the policy gets applied. Sometimes you find the error there because you missed some letters, got a faulty registry key or "path" or copy&pasted an ADM template which contains a line break within the KEYNAME-section. This all causes the keyname to become "corrupt".

If the keyname is correct, check whether you have the right valuename and the right value type. Don't confuse REG_SZ and REG_DWORDs.

If this all does not help, make sure the application you want to handle with your ADM template does look at the registry key you're altering. It makes so sense altering a registry key if no application does look at it and recognizes you made changes to it.

Ending words

Final thoughts

So this paper provided a look into how to create custom ADM templates and therefore how to deploy registry values to client machines. We saw a lot of examples that (hopefully) made the default procedures of creating own templates much clearer and provided a deeper insight into the cryptic ADM template files.

Any suggestions on how this paper could be improved, feel free to write an email to: whitepapers@frickelsoft.net

Further Reading

Of course there are a lot of good links about how to create custom ADM templates. Here are a few:

"Language Reference for Administrative Template Files", http://technet2.microsoft.com/windowsserver/en/library/9687f84f-8cd0-4cde-81c0-53c65a42a0c31033.mspx?mfr=true

"Windows System Policy Editor: Chapter 8: Creating a Custom Template", http://www.oreilly.de/catalog/winsyspe/chapter/ch08.html

"Using Administrative Template Files with Registry-Based Group Policy", http://www.microsoft.com/technet/prodtechnol/windowsserver2003/technologies/management/gp/admtgp.mspx

About the author

Florian Frommherz is a 23-year-old student for information technology (IT) engineering. He's studying at the University of Cooperative Education – Berufsakademie Loerrach, where both practical and theoretical aspects of IT are taught. Besides the technical aspects taught at BA Loerrach, he works at ControlTech Engineering AG (CTE) in Liestal/Switzerland to get practical experience. His main fields are writing small applications in .NET and administering the internal corporate network. Since July 2007, he holds the Microsoft MVP award for "Windows Server – Group Policy" for his volunteer work in Microsoft newsgroups. You can read more of his work and more about Group Policy on his blog: http://www.frickelsoft.net/blog

Sources

[1] “Windows Registry”, http://en.wikipedia.org/wiki/Windows_registry, The Wiki Project, July, 7 2007.

[2] “Language Reference for Administrative Template files”, http://technet2.microsoft.com/windowsserver/en/library/9687f84f-8cd0-4cde-81c0-53c65a42a0c31033.mspx?mfr=true, Microsoft Corp., October, 16 2007.

[3] “ConTEXT Programmers Editor”, http://www.context.cx, November 2nd, 2007

[4] “ConTEXT Programmers Editor – download highlighters”, http://www.context.cx/component/option,com_docman/task,cat_view/gid,72/Itemid,48/, November 2nd, 2007