



Operating System

Windows 2000 Active Directory Display Specifiers

White Paper

Abstract

This white paper introduces Microsoft® Windows® 2000 Active Directory Display Specifiers. Display Specifiers are objects that hold Active Directory user interface (UI) information and provide a flexible UI mechanism to meet the needs of the various user groups in the distributed network.

© 1999 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.

The BackOffice logo, Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

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

*Microsoft Corporation • One Microsoft Way • Redmond, WA 98052-6399 • USA
1999*

CONTENTS

INTRODUCTION	1
UI ELEMENT SPECIFICATION STORED IN THE DIRECTORY	2
Property Pages	2
Context Menus	3
Class Icons	4
Object Creation Wizard	4
Class and Attribute Names	5
Viewing Containers as Leaf Nodes	5
DISPLAY SPECIFIER CONTAINER	6
Caching Display Information	6
LOCALIZATION	7
CUSTOMER MODIFICATIONS	8
New Classes	8
Modifying Existing Classes	8
EXAMPLE DISPLAY SPECIFIER	9
FOR MORE INFORMATION	10

INTRODUCTION

The Active Directory is a sophisticated, adaptive directory service that allows a high degree of customer modification to meet specific business and organizational needs. For example, existing object classes can be modified by changing the set of contained attributes. New attributes can be created, and to some extent, existing attributes can be removed.

New object classes can be created, inheriting the characteristics of one or more existing classes, and new attributes can be added. These changes to the Active Directory are relatively easy to make. Consequently, the user interface (UI) used to access these objects needs to be relatively easy to extend and customize.

Administrators and end users have different requirements for user interfaces. Many properties and actions have no meaning to end users and yet are important for administrators. Moreover, the Active Directory has a fine-grained security model that allows permissions to be defined down to an individual attribute level and administration tasks to be delegated to different users. Thus, the Active Directory supports a UI that adapts to meet the needs of administrators and end users, extends to support modifications to the schema, and reflects the fine-grained security model.

The administrative UI is presented through different Microsoft Management Console (MMC) snap-ins—specifically the Active Directory Manager, Active Directory Sites and Services Manager, Active Directory Tree Manager, and Active Directory Schema Manager.

The end user, however, sees the directory through the Windows shell. Users can browse for objects stored in the Active Directory from either the Network Neighborhood on the Desktop or through the Find dialog boxes available in the Start menu.

Although their user interfaces and their experience with the Active Directory will differ, both administrators and users will require that directory service objects be displayed in the user interface. Therefore, a flexible UI mechanism is needed to meet the needs of the various user groups while still meeting the more general UI goals of localization, extensibility, and ease of customer modification.

UI ELEMENT SPECIFICATION STORED IN THE DIRECTORY

User interface information is stored in the Active Directory. This UI specification is done at the granularity of each object class, and object classes are defined in that portion of the directory known as the *schema*. These objects are called *schema class objects*. Each schema class object can have UI display specification information associated uniquely with it.

The Windows 2000 Active Directory can specify various UI elements on a per-class basis. These elements are property pages, context menus, icons, creation wizards, and localized class and attribute names.

The display specification system stores information for property sheets, context menus, icons, creation wizards, and localized class and attribute names. The display specification system uses this information to form different user interfaces for administrators and end users—one set of elements, such as property pages, context menus, and so on, can be associated with administrative applications, while a different set of elements can be associated with end-user applications.

This UI information is stored in an Active Directory object called a *Display-Specifier*. Each Display-Specifier object contains attributes describing the various UI elements for the specific user interface to which it pertains. Each Display-Specifier object is stored in a Display Specifiers container that corresponds to each locale supported by Windows 2000.

Property Pages

Each property page for a Display-Specifier object is a Component Object Model (COM) object. The description of a COM object (its universally unique identifier, or UUID) is stored in an attribute on a Display-Specifier object. These property page attributes are multivalued, with each element holding the description for a single COM object. These property page attributes are called Admin-Property-Pages and Shell-Property-Pages.

A class UUID that is registered with the system and activated using the standard COM instance creation methods names a COM property page. The object must implement two shell interfaces: **IShellExtInit** and **IShellPropSheetExt**.

The description of a property page COM object is stored in the Display-Specifier attribute as a string with the following format:

```
<order-number>, <CLSID>, [optional data]
```

where

- The order-number determines the page's position on the sheet. Order-numbers are sorted using a signed comparison so that there is no prescribed starting position and there can be gaps in the sequence.
- The CLSID, or class identifier, must be a string representation of a UUID enclosed in curly braces.

-
- The optional data is passed to the COM object by the **IShellExtInit::Initialize** data object. The clipboard data format is currently named `CFSTR_DSPPROPERTYPAGEINFO`, but that name is subject to change before the final shipment of Windows 2000.

Each COM object can implement more than one property page. One possible use of the optional data is to name the pages to display. This gives implementers a great deal of flexibility in determining the number of COM objects needed to create their required pages.

Context Menus

Context menu items may be either COM objects, which are activated by the standard COM instance creation methods, or an application that is invoked by the standard **ShellExec** function. Similar to property pages, context menu attributes are multivalued, with each element holding the description for a single COM object or application.

These attributes are called Admin-Context-Menu and Shell-Context-Menu. In addition, there is an attribute, Context-Menu, for menu items that are common to both administrative and end-user UIs.

A class UUID registered with the system and activated by the standard COM instance creation methods names a COM context menu. The object must implement two shell interfaces: **IShellExtInit** and **IContextMenu**.

The description of a context menu COM object is stored in the Display-Specifier context menu attribute as a string with the following format:

```
<order-number>,<CLSID>,[optional data]
```

where

- The order-number determines the context menu item's position on the context menu. Order-numbers are sorted using a signed comparison so that there is no prescribed starting position and there can be gaps in the sequence.
- The CLSID, or class identifier, must be a string representation of a UUID enclosed in curly braces.
- The COM object must implement the **IContextMenu** interface.
- The optional data is passed to the COM object by the **IShellExtInit::Initialize** data object.

The description of an application is stored in the **Display-Specifier** attribute as a string with the following format:

```
<order-number>,<context menu name>,<program name>
```

where

- The order-number determines the context menu item's position on the context menu. Order-numbers are sorted using a signed comparison so that there is no prescribed starting position and there can be gaps in the sequence.
- The context menu name is the text of the menu item that appears in the context menu.
- The program name is the application that is executed by the snap-in. Either the full path must be specified or the application must be in the search path.
- The selected object's distinguished name and class are passed as the first and second arguments respectively.

Class Icons

The iconic images used to represent a class object can be read from the Display Specifier. Moreover, each class can store multiple icon states. For example, a folder class can have bitmaps for the open, closed, and disabled states. The current implementation allows up to 16 different icon states per class.

The attribute is named Class-Icon and can be specified in one of two ways.

```
<state>,<ICO-file-name>
```

or

```
<state>,<DLL-name>,<resource-ID>
```

where

- The state is an integer with a value between 0 and 15. The value 0 is defined to be the default or closed state of the icon. The value 1 is defined to be the open state of the icon. The value 2 is the disabled state. The other values are application-defined.
- The ICO file name or DLL-name must be the name of a file in the local computer's file search path.
- The resource-ID is a 0-based index into the DLL's resource fork list of icons.

Object Creation Wizard

Creating new instances of an object invokes the object creation wizard. Each class of object may specify the use of a specific creation wizard, or it may use a generic creation wizard. For well known classes (for example, **user** or **organizationalUnit**), the Active Directory Manager snap-in provides a standard set of creation wizards.

There are two methods for extending creation wizards: either completely replacing an existing wizard or extending an existing wizard.

The existing wizard is replaced through the use of a *primary extension*. A primary extension provides the first set of pages and is hosted in the same way as native pages. It also supports the extensibility mechanism so that other creation wizard extensions can be invoked.

Existing wizards can be extended with creation wizard extensions, which add additional pages following those of the native or primary extension.

In both cases, the extension UI has to be implemented as a COM object, and it must support the **IDsAdminWizExt** interface. A class UUID registered with the system and activated by the standard COM instance creation methods names both a creation wizard and a creation wizard extension.

The description of a creation wizard COM object is stored in the single-valued **Creation-Wizard** attribute as a string with the following format:

```
<CLSID>
```

where the CLSID, or class identifier, must be a string representation of a UUID enclosed in curly braces.

The description of a creation wizard extension COM object is stored in the multivalued **Create-Wizard-Ext** attribute as a string with the following format:

```
<order-number>, <CLSID>
```

where

- The order-number determines the extension's position in the wizard. Order-numbers are sorted using a signed comparison so that there is no prescribed starting position and there can be gaps in the sequence.
- The CLSID, or class identifier, must be a string representation of a UUID enclosed in curly braces.

Unlike property page COM objects, a creation wizard COM object can only support a single creation wizard.

Class and Attribute Names

Each object class may also have a class display name, and each attribute of that class may have an attribute display name. The class display name is stored in the **Class-Display-Name** attribute as a single-valued Unicode string. The attribute display names are stored in the **Attribute-Display-Name** attribute as multivalued Unicode strings, with each element consisting of a comma-delimited name pair. (The first name is the attribute LDAP Display Name, which is followed by a comma, and then the corresponding UI display name.)

Viewing Containers as Leaf Nodes

Potentially, any Active Directory object can be a container of other objects. This can clutter the UI, so it is possible to declare that a specific class be displayed as a leaf element by default. The **Treat-As-Leaf** attribute holds a boolean value that, if True, indicates that the objects of the class should be treated as leaf elements.

DISPLAY SPECIFIER CONTAINER

The **Configuration** container stores the **DisplaySpecifiers** container, which in turn stores containers that correspond to each locale. These locale containers are named using the hex representation of that locale's LCID. Thus the US/English locale's container is named 409, the German locale's container is named 407, the Japanese locale's container is named 411, and so on.

Each locale container stores objects of the class **Display-Specifier**. Display-Specifier objects are named by appending the string “-Display” to the LDAP Display Name of the class object.

For example, the user class has a corresponding Display-Specifier object called “user-Display.” Thus, when you bind to an object of a particular class, you look up the Display-Specifier object based on the same name as the class and within the container for the current locale.

Caching Display Information

The display information is fetched when a new class is encountered and the results saved for displaying other objects of the same class. The cache is per-process and non-persistent.

LOCALIZATION

Each locale container holds Display-Specifier objects that have been localized for that locale. The Active Directory UI applications first looks in a locale container named after the locale identifier for the current user's session. If a folder of that name is not found, the US-English locale is used.

COM objects can be localized either by having a separate binary for each language or by having multiple language resources in a single binary.

Similarly, the class and attribute display names are translated. Different icons can be specified for each locale.

CUSTOMER MODIFICATIONS

Users of the Active Directory can customize it to suit their unique requirements. The user interface can also be changed to suit their needs. The Active Directory permits the schema to be modified by creating new classes and attributes or modifying existing classes. Display Specifiers can be modified to reflect the new user interface elements that schema modifications require.

New Classes

A Display-Specifier object is created for the new class. If multiple locales are present, new Display-Specifier objects are created for each supported locale. If the new class is derived from a parent class and the parent class already has an acceptable UI, that UI is specified and additional pages, menu items, and display names can be added for the new attributes. Otherwise, a completely new UI is created and specified. In either case, a new icon and class display name can be used.

Modifying Existing Classes

New attributes can be added to an existing class. New UI components (pages, menu items, and attribute display names) can be added, or the existing UI replaced. It is also possible to design new property pages that expose fewer attributes of a class and to create context menus with fewer actions.

EXAMPLE DISPLAY SPECIFIER

Here is an example of an actual display specifier in the Windows 2000 Active Directory.

```
[group-Display]
objectClass = displaySpecifier
ObjectCategory = Display-Specifier
cn = group-Display
adminPropertyPages = 1,{6dfe6489-a212-11d0-bcd5-00c04fd8d5b6}
adminPropertyPages = 2,{6dfe648b-a212-11d0-bcd5-00c04fd8d5b6}
adminPropertyPages = 3,{6dfe6488-a212-11d0-bcd5-00c04fd8d5b6}
adminPropertyPages = 4,{4E40F770-369C-11d0-8922-00A024AB2DBE}
shellPropertyPages = 1,{f5d121ee-c8ac-11d0-bcdb-00c04fd8d5b6}
shellPropertyPages = 2,{dde2c5e9-c8ae-11d0-bcdb-00c04fd8d5b6}
contextMenu = 0,{62AE1F9A-126A-11D0-A14B-0800361B1103}
adminContextMenu = 1,{08eb4fa6-6ffd-11d1-b0e0-00c04fd8dca6}
classDisplayName = Group
attributeDisplayNames = cn,Name
attributeDisplayNames = c,Country Abbreviation
attributeDisplayNames = description,Description
attributeDisplayNames = distinguishedName,X500 Distinguished Name
attributeDisplayNames = 1,City
attributeDisplayNames = managedBy,Managed By
attributeDisplayNames = member,Members
attributeDisplayNames = notes,Notes
attributeDisplayNames = physicalDeliveryOfficeName,Delivery Office
attributeDisplayNames = url,Web Page Address
treatAsLeaf=True
```

FOR MORE
INFORMATION

For the latest information on Windows NT Server, visit the Web site at <http://www.microsoft.com/ntserver> and the Windows NT Server Forum on the Microsoft Network (GO WORD: MSNTS).

For detailed programming information about COM, the Microsoft Management Console (MMC), and the Active Directory, see the Microsoft Windows Platform SDK.