

# Centrify Zero Trust Privilege Services: Authentication Service, Privilege Elevation Service, and Audit and Monitoring Service

*Windows API Programmer's Guide*

December 2019 (release 19.9)

Centrify Corporation





## Legal Notice

This document and the software described in this document are furnished under and are subject to the terms of a license agreement or a non-disclosure agreement. Except as expressly set forth in such license agreement or non-disclosure agreement, Centrifly Corporation provides this document and the software described in this document “as is” without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose. Some states do not allow disclaimers of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

This document and the software described in this document may not be lent, sold, or given away without the prior written permission of Centrifly Corporation, except as otherwise permitted by law. Except as expressly set forth in such license agreement or non-disclosure agreement, no part of this document or the software described in this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, or otherwise, without the prior written consent of Centrifly Corporation. Some companies, names, and data in this document are used for illustration purposes and may not represent real companies, individuals, or data.

This document could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein. These changes may be incorporated in new editions of this document. Centrifly Corporation may make improvements in or changes to the software described in this document at any time.

© 2004-2019 Centrifly Corporation. All rights reserved. Portions of Centrifly software are derived from third party or open source software. Copyright and legal notices for these sources are listed separately in the Acknowledgements.txt file included with the software.

U.S. Government Restricted Rights: If the software and documentation are being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), in accordance with 48 C.F.R. 227.7202-4 (for Department of Defense (DOD) acquisitions) and 48 C.F.R. 2.101 and 12.212 (for non-DOD acquisitions), the government’s rights in the software and documentation, including its rights to use, modify, reproduce, release, perform, display or disclose the software or documentation, will be subject in all respects to the commercial license rights and restrictions provided in the license agreement.

Centrifly, DirectControl, DirectAuthorize, DirectAudit, DirectSecure, DirectControl Express, Centrifly for Mobile, Centrifly for SaaS, DirectManage, Centrifly Express, DirectManage Express, Centrifly Suite, Centrifly User Suite, Centrifly Identity Service, Centrifly Privilege Service and Centrifly Server Suite are registered trademarks of Centrifly Corporation in the United States and other countries. Microsoft, Active Directory, Windows, and Windows Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries.

Centrifly software is protected by U.S. Patents 7,591,005; 8,024,360; 8,321,523; 9,015,103; 9,112,846; 9,197,670; 9,442,962 and 9,378,391.

The names of any other companies and products mentioned in this document may be the trademarks or registered trademarks of their respective owners. Unless otherwise noted, all of the names used as examples of companies, organizations, domain names, people and events herein are fictitious. No association with any real company, organization, domain name, person, or event is intended or should be inferred.



# Contents

<b>About this guide</b> .....	<b>7</b>
Intended audience .....	7
Using this guide .....	8
Compatibility and limitations .....	9
Documentation conventions .....	9
Finding more information about Centrify products .....	10
Product names .....	10
Contacting Centrify .....	12
Getting additional support .....	12
<b>Developing programs using Centrify objects</b> .....	<b>14</b>
Introduction to the development platform .....	14
Available tools for Windows developers .....	15
Available tools for UNIX developers .....	16
<b>Installing the Centrify SDK package</b> .....	<b>17</b>
Downloading a standalone SDK package .....	17
Installing the standalone SDK package .....	18
<b>Overview of the Centrify Windows API object model</b> .....	<b>19</b>
How the Centrify Windows API relies on COM interfaces .....	19
Administrative tasks you can perform .....	21
Centrify-specific objects classes .....	21
Creating objects in the proper order .....	23
Creating the top-level Cims object .....	27
Working with NIS maps .....	27
Writing scripts that use Centrify Windows API calls .....	28



<b>Centrify object reference .....</b>	<b>34</b>
AzRoleAssignment .....	36
Cims .....	39
Command .....	65
Commands .....	74
Computer .....	75
ComputerGroupUnixProfiles .....	92
ComputerRole .....	93
ComputerRoles .....	109
Computers .....	110
ComputerUserUnixProfiles .....	111
CustomAttributeContainer .....	113
CustomAttributes .....	114
CustomAttribute .....	115
Group .....	115
GroupUnixProfile .....	126
GroupUnixProfiles .....	143
HierarchicalGroup .....	146
HierarchicalUser .....	158
HierarchicalZone .....	184
HierarchicalZoneComputer .....	249
HzRoleAssignment .....	289
InheritedRoleAsg .....	291
Key .....	295
Keys .....	301
License .....	304
Licenses .....	309
LicensesCollection .....	315
MzRoleAssignment .....	320



NetworkAccess .....	322
NetworkAccesses .....	327
Pam .....	327
Pams .....	330
Right .....	331
Role .....	334
RoleAssignment .....	354
RoleAssignments .....	362
Roles .....	363
Ssh .....	364
Sshs .....	365
User .....	366
UserUnixProfile .....	378
UserUnixProfiles .....	394
WindowsApplication .....	397
WindowsApplicationCriteria .....	404
WindowsApplications .....	420
WindowsDesktop .....	421
WindowsDesktops .....	425
WindowsUser .....	426
WindowsUsers .....	429
Zone .....	430
Entry .....	489
Map .....	497
Store .....	510
GroupInfo .....	519
GroupInfos .....	528
GroupMember .....	531
GroupMembers .....	532



UserInfo .....	535
UserInfos .....	545
<b>Data storage for Centrify zones .....</b>	<b>548</b>
Basic requirements .....	548
Schemas and zones .....	549
The logical data model for objects .....	550
Differences between types of zones .....	554
Classic Centrify zones (2.x, 3.x, 4.x) .....	555
Classic RFC 2307 zones (3.x, 4.x) .....	564
Classic SFU-compliant zones (version 3.5) .....	571
Classic SFU-compliant zones (version 4.0) .....	576
Hierarchical Centrify zones (5.x) .....	578
Using commands and scripts to perform tasks .....	587
<b>Adding users in a one-way trust environment .....</b>	<b>592</b>
<b>Reading and setting timebox values .....</b>	<b>594</b>
Hex string .....	594
Hour mapping .....	594
Day mapping .....	596



# About this guide

The *Windows API Programmer's Guide*, object reference help, and sample scripts are key components of the Centrify Software Development Kit (SDK). This *Windows API Programmer's Guide* describes the basic object model that Centrify software components use to manage the Windows and UNIX-specific properties for users, groups, computers, zones, and Network Information Services (NIS) maps.

This guide and the object reference help provide complete reference information for the application programming interfaces (APIs) that you can use to automate the provisioning of zones, users, and groups into Microsoft Active Directory with custom scripts or applications. These programming interfaces are installed automatically on any computer where you install the Access Manager console.

## Intended audience

This document provides reference information and examples for programmers who plan to use the Centrify SDK to develop programs for Windows and UNIX environments. It includes information for managing both Windows and UNIX computers and for managing the Active Directory data associated with UNIX users, groups, computers, and network maps. Much of the information in this guide is primarily intended for developers writing programs to provision UNIX users and groups into an Active Directory environment.

Before using the Centrify SDK to develop programs that run on Windows, you should have experience programming using Microsoft .NET or COM-enabled languages. To develop programs to run on UNIX computers, you must also understand LDAP commands and scripting. You should also understand the structure of Active Directory, including the Active Directory schema your organization is using, and how to use Active Directory MMC snap-ins.

In addition to programming skills, you should be familiar with Centrify Authentication Service, Privilege Elevation Service, and Audit & Monitoring Service architecture, terms, and concepts, and understand how to perform administrative tasks using Centrify software and the UNIX platforms you



support. This guide assumes you are familiar with the information in the *Administrator's Guide for Linux and UNIX*.

## Using this guide

This guide discusses the Centrify SDK COM objects and provides detailed information about how Centrify-specific information is stored in Active Directory. This information is intended to help you develop programs for creating and populating zones and provisioning users, groups, and computers on Windows or UNIX computers. Depending on your environment or interests, you may want to read portions of this guide selectively. The guide provides the following information:

- **Developing programs using Centrify objects** provides an introduction to the Centrify SDK for application developers.
- **Installing the Centrify SDK package** describes the Centrify SDK components and how to install the SDK as a standalone package on a computer.
- **Overview of the Centrify Windows API object model** provides an overview of the Centrify objects you can use to manipulate and manage information stored in Active Directory. It includes simple examples of using the API in Windows-based scripting languages.
- **Centrify object reference** provides a reference to the methods and properties that compose the API. Detailed reference information for these objects is also available in the compiled help (.CHM) file included with the SDK package.
- **Data storage for Centrify zones** describes the differences in how data is stored for different types of zones and Active Directory schema options. This chapter also provides simple examples that illustrate how to use LDAP command line programs to access and manipulate Centrify data on UNIX computers. If you plan to develop programs to manage and manipulate data from a UNIX computer, you should go directly to this chapter.
- **Adding users in a one-way trust environment** explains how to add a user in a one-way trust environment.
- **Reading and setting timebox values** explains the format of a timebox value. The timebox field in a role object defines when a role's rights are in effect.



## Compatibility and limitations

The information in this guide is intended for use with the current version of Centrify software. Although intended to be accurate and up-to-date, interfaces are subject to change without notice and may become incompatible or obsolete when a newer version of the software is released. In general, application programming interfaces are also intended to be backward-compatible, but are not guaranteed to work with older versions of the software. Because the Centrify APIs are subject to change, enhancement, or replacement, the information in this guide can also become incomplete, obsolete, or unsupported in future versions. If you are unsure whether this guide is appropriate for the version of the software you have installed, you can consult the Centrify website or Centrify Support to find out if another version of this guide is available.

## Documentation conventions

The following conventions are used in Centrify documentation:

- Fixed-width font is used for sample code, program names, program output, file names, and commands that you type at the command line. When *italicized*, this font indicates variables. Square brackets ([ ]) indicate optional command-line arguments.
- **Bold** text is used to emphasize commands or key command results; buttons or user interface text; and new terms.
- *Italics* are used for book titles and to emphasize specific words or terms. In fixed-width font, italics indicate variable values.
- Standalone software packages include version and architecture information in the file name. Full file names are not documented in this guide. For complete file names for the software packages you want to install, see the distribution media.
- For simplicity, UNIX is used to refer to all supported versions of the UNIX and Linux operating systems. Some parameters can also be used on Mac OS X computers.



## Finding more information about Centrify products

Centrify provides extensive documentation targeted for specific audiences, functional roles, or topics of interest. If you want to learn more about Centrify and Centrify products and features, start by visiting the [Centrify website](#). From the Centrify website, you can download data sheets and evaluation software, view video demonstrations and technical presentations about Centrify products, and get the latest news about upcoming events and webinars.

For access to documentation for all Centrify products and services, visit the [Centrify documentation portal](#) at [docs.centrify.com](https://docs.centrify.com). From the Centrify documentation portal, you can always view or download the most up-to-date version of this guide and all other product documentation.

For details about supported platforms, please consult the release notes.

For the most up to date list of known issues, please login to the Customer Support Portal at <http://www.centrify.com/support> and refer to Knowledge Base articles for any known issues with the release.

## Product names

Over the years we've made some changes to some of our product offerings and features and some of these previous product names still exist in some areas. Our current product offerings include the following services:

<b>Current Overall Product Name</b>	<b>Current Services Available</b>
Centrify Zero Trust Privilege Services	Privileged Access Service
	Gateway Session Audit and Monitoring
	Authentication Service
	Privilege Elevation Service
	Audit and Monitoring Service
	Privilege Threat Analytics Service

Whether you're a long-time or new customer, here are some quick summaries of which features belong to which current product offerings:



Previous Product Offering	Previous Product Offering	Description	Current Product Offering
	Centrify Privileged Service (CPS)		Privileged Access Service
DirectControl (DC)			Authentication Service
DirectAuthorize (DZ or DZwin)			Privilege Elevation Service
DirectAudit (DA)			Audit and Monitoring Service
	Infrastructure Services		Privileged Access Service, Authentication Service, Privilege Elevation Service, Audit and Monitoring Service, and Privilege Threat Analytics Service
DirectManage (DM)	Management Services	Consoles that are used by all 3 services: Authentication Service, Privilege Elevation Service, and Audit and Monitoring Service	
DirectSecure (DS)	Isolation and Encryption Service		Still supported but no longer being developed or updated
	User Analytics Service		Privilege Threat Analytics Service

Depending on when you purchased a Centrify product offering, you may have purchased one of the following product bundles:

Previous Product Bundle	Previous Product Bundle	Current Product Bundle	Services Included	Description
		Centrify Zero Trust Privilege Services Core Edition	Privileged Access Service and Gateway Session Audit and Monitoring	
Centrify Server	Centrify Infrastructure	Centrify	Privileged Access Service, Authentication Service, and	



Previous Product Bundle	Previous Product Bundle	Current Product Bundle	Services Included	Description
Suite Standard Edition	Services Standard Edition	Zero Trust Privilege Services Standard Edition	Privilege Elevation Service	
Centrify Server Suite Enterprise Edition	Centrify Infrastructure Services Enterprise Edition	Centrify Zero Trust Privilege Services Enterprise Edition	Privileged Access Service, Authentication Service, Privilege Elevation Service, Audit and Monitoring Service (includes Gateway Session Audit and Monitoring)	
Centrify Server Suite Platinum Edition				Discontinued bundle that included DirectControl, DirectAuthorize, DirectManage, DirectAudit, and DirectSecure

## Contacting Centrify

You can contact Centrify by visiting our website, [www.centrify.com](http://www.centrify.com). On the website, you can find information about Centrify office locations worldwide, email and phone numbers for contacting Centrify sales, and links for following Centrify on social media. If you have questions or comments, we look forward to hearing from you.

## Getting additional support

If you have a Centrify account, click Support on the Centrify website to log on and access the [Centrify Technical Support Portal](#). From the support portal, you can search knowledge base articles, open and view support cases, download software, and access other resources.



To connect with other Centrify users, ask questions, or share information, visit the [Centrify Community](#) website to check in on customer forums, read the latest blog posts, view how-to videos, or exchange ideas with members of the community.



# Developing programs using Centrify objects

The Centrify Software Development Kit (SDK) consists of the following:

- Application programming interfaces that packaged in a dynamic link library (.DLL).
- A compiled help file that includes complete object reference information.
- Sample scripts in multiple languages.
- Supporting documentation and reference information for UNIX developers.

On Windows computers, you can use the API to develop your own custom applications that access or modify Centrify-specific data in Active Directory.

As part of the Centrify SDK, this guide also provides detailed information about the underlying Centrify data storage model and how attributes managed by Centrify are stored in Active Directory. This information is critical for developing programs that access or modify Centrify data but are run on UNIX computers.

## Introduction to the development platform

You can use the Windows API to develop programs that manage UNIX user, group, and computer profiles; zones and zone properties; and NIS maps and NIS map entries. The methods and properties that make up the API enable you to access, create, modify, and remove information stored in Active Directory. Although you can use the Windows API to manage all of the UNIX information stored in Active Directory, including UNIX profile attributes and computer accounts, the API does not run on UNIX computers.

If you want to develop programs that run on UNIX computers to access data that's stored in Active Directory, you can use the ADEdit program (`adedit`) or the command line programs included with the Centrify UNIX agent to perform



queries and updates. For example, you can use ADEdit commands in custom scripts to create zones and add, update, or remove users and groups. For detailed information about using ADEdit, see the *ADEdit Command Reference and Scripting Guide*.

You can also use OpenLDAP commands to manipulate data in Active Directory directly. The key to writing programs that use OpenLDAP or other commands is understanding how the data is stored in Active Directory and the command line options supported for each of the commands you want to use. For information about using command line programs, see the man page for the corresponding program.

Depending on the task you want to perform and the development platform you want to use, you can write scripts that manage Centrify data using either the Windows API or UNIX command line programs. For example, you can perform most provisioning-related tasks using calls to the objects, methods, and properties in the Windows API, or using common LDAP commands on UNIX.

## Available tools for Windows developers

If you plan to develop programs that run on Windows computers, the Centrify SDK includes the following:

- A library of commands for access control and privilege management that run in Windows PowerShell.
- Sample scripts that use the Centrify PowerShell cmdlets to illustrate common administrative tasks.
- Dynamic link libraries that expose interfaces for working with Centrify objects and attributes stored in Active Directory.
- Sample scripts that illustrate adding and removing users, groups, and zones in VBScript, PowerShell, and .NET (C#) languages.
- An overview of the programming interface architecture, its relationship to the Access Manager console, and the object properties and methods available.
- Reference information for all object properties and methods.

For more information about using Windows PowerShell cmdlets, see the *Access Control and Privilege Management Scripting Guide*.



For more information about developing COM- or .NET-based applications, see [Overview of the Centrify Windows API object model](#) and [Centrify object reference](#)

For a more detailed understanding of how Centrify-specific data is stored by zone type, see [Data storage for Centrify zones](#)

## Available tools for UNIX developers

If you plan to develop programs that run on UNIX computers, the Centrify SDK includes the following:

- The ADEdit command line application and library of procedures for scripting access control and privilege management tasks.
- Sample scripts written in ADEdit that illustrate common administrative tasks.
- Detailed information about how data is stored in Active Directory and how data storage differs by zone type.
- Examples that illustrate how to use OpenLDAP commands and other command line tools to perform common administrative tasks.

For detailed information about using ADEdit, see the *ADEdit Command Reference and Scripting Guide*. For more information about developing scripts that use LDAP commands, see [Data storage for Centrify zones](#)



# Installing the Centrify SDK package

You can install the Centrify SDK from the Centrify Management Services setup program or as a separate package. It includes the access control and privilege management application programming interface (API), sample scripts, and documentation for performing common administrative tasks using Windows scripting languages. This chapter describes how to install the standalone SDK package on a Windows computer.

## Downloading a standalone SDK package

The Windows-based programming interfaces are defined in dynamic link libraries that can be installed on any computer where you install other Windows-based components. The COM-based Windows API is installed automatically on any computer where you install Access Manager. You can also download these libraries separately, along with sample scripts and documentation, onto computers where Access Manager is not installed.

The .NET assemblies that are installed with Access Manager and make up the Windows API can be called from scripts written in any COM-compliant language, such as VBScript or PowerShell. Sample scripts are provided in multiple languages as examples of how to perform common administrative tasks using calls to the Windows API.

You can download the Centrify SDK as a separate software package from the Centrify Download Center under **Software Development Kits**. The standalone package is a single compressed folder for Windows 64-bit computers. However, you must obtain an unlocking code or license key from your Centrify sales representative to access the SDK.



## Installing the standalone SDK package

After you have downloaded the compressed file to your computer, you can extract the files and run the setup program to install the SDK libraries and sample scripts.

### To run the standalone setup program:

1. Select the downloaded file, right-click, then select **Extract All** to extract the SDK files to a folder.
2. Double-click the downloaded executable to start the setup program.  
For example, for the 64-bit version of the 5.1.3 SDK, double click the `centrifDC_SDK-5.1.3-win64.exe` file.
3. At the Welcome page, click **Next**.
4. Type your name and company, select who should be able to use the application on the computer, then click **Next**.
5. Accept the default location or click **Browse** to choose a different location, then click **Next**.
6. Select the components you want to install, then click **Next**.
7. Click **Finish** to complete the installation.



# Overview of the Centrify Windows API object model

This chapter provides an overview of the architecture and capabilities of the object library exposed in the Centrify Windows API included in the SDK and how you can use the objects in applications to access and manage Centrify data on Windows computers. It includes a discussion of the Centrify framework classes, the order in which objects are created, and examples of how to use the programming interfaces in scripts written in VBScript, PowerShell, and .NET languages.

## How the Centrify Windows API relies on COM interfaces

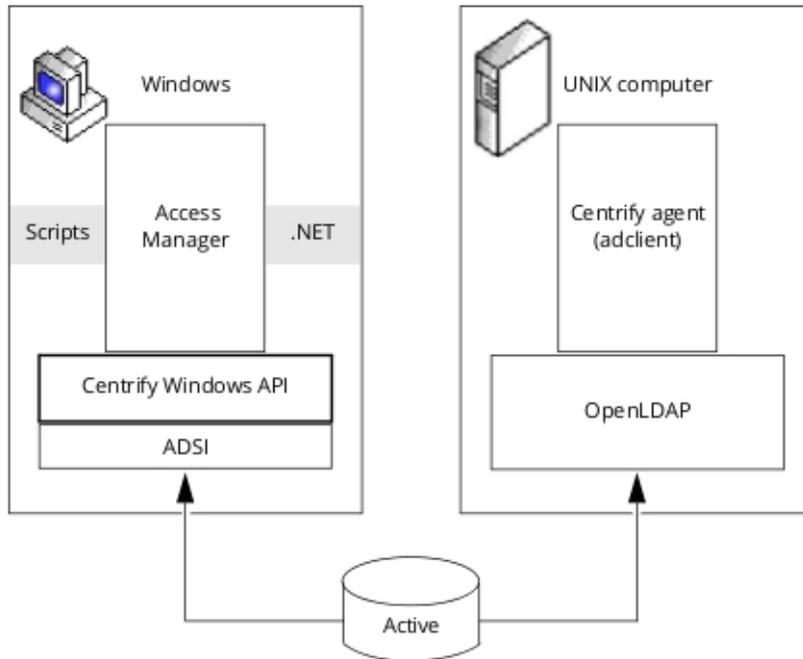
On Windows computers, the Centrify API supports the Component Object Model (COM) interface. The Component Object Model (COM) interface enables you to create objects that can interact with Active Directory or be used in other applications. These are re-usable objects that can provide access to all of the Centrify data stored in Active Directory. The objects can be used in any program written in .NET or COM-enabled languages. You can, therefore, create or modify applications to use these objects in COM-aware languages such as VBScript and PowerShell or .NET-compliant languages such as C#. The object model used to access the data is the same, but the specific syntax required depends on the programming language you choose to use.

The objects that make up the Centrify Windows API rely on the underlying interfaces provided by Microsoft's Active Directory Service Interfaces (ADSI). ADSI provides the base-level functions that permit applications to read and write data in Active Directory. The purpose of the Centrify Windows API is to



provide a higher level of abstraction for performing Centrify-specific tasks than would be available if you were to call ADSI functions directly.

The following figure illustrates how the Centrify Windows API provides a layer of abstraction between the raw ADSI functions and the Access Manager console and other applications.



The Active Directory schema defines how all of the objects and attributes in the database are stored. When you add Centrify data to the Active Directory database, how that data is stored depends on the Active Directory schema you have installed. The Centrify Windows API, however, provides a logical view of the data, eliminating the need to know the details of how data is stored in different schemas when performing common administrative tasks. The Centrify Windows API also provides a simpler interface for accessing the well-defined set of UNIX objects that must be operated on than that offered by the general purpose ADSI. In fact, when you perform administrative tasks with the Access Manager console MMC snap-in, the console uses the same Centrify Windows API objects documented in this guide to manipulate the data.

Therefore, with the Centrify Windows API and any commonly-used Windows programming language, you can write scripts or programs that perform a wide range of tasks using Centrify data, including programs that automatically create and manage Centrify zones or update user, group, or computer properties.

**Note:** You can use ADSI directly instead of using the Centrify Windows API, if you prefer. For more detailed information about the objects



and attributes used in Active Directory when different schemas are used, see [Data storage for Centrify zones](#)

## Administrative tasks you can perform

Using the Centrify Windows API, you can perform a wide range of common administrative tasks, including the following:

- Add, modify, and delete zones, including hierarchical zones.
- Add, modify, and delete users within zones.
- Add, modify, and delete groups in zones.
- Add, modify, and delete computers in zones.
- Create, modify, and delete rights, roles, and role assignments.
- Check on licenses and keys.
- Create, modify, and delete NIS maps and NIS map entries.

For examples of performing common activities such as these using Centrify objects, see the sample scripts and help included with the software package.

Although you can use the Centrify Windows API to perform many common administrative tasks programmatically, you cannot create new Active Directory users or groups directly. To create new Active Directory user or group objects, you must use the underlying Active Directory Service Interfaces (ADSI) rather than the abstracted interface provided by the Centrify Windows API. To learn how to create user or group objects programmatically or perform other tasks that are not provided by the Centrify Windows API, such as changing access rights, refer to Microsoft's ADSI documentation.

## Centrify-specific objects classes

The Centrify Windows API consists of several common, interdependent classes that correspond with the core elements of Centrify-managed data, such as computers, users, groups, and zones. These basic classes provide properties, methods, and attributes that you can manipulate in programs and scripts to set or retrieve data.

The following table lists the classes that compose the Centrify Windows API.



<b>Class</b>	<b>Description</b>
<code>AZRoleAssignment</code>	Represents a computer-role assignment.
<code>Cims</code>	Initiates interaction with Active Directory. This top-level class connects to Active Directory and prepares the Active Directory domain and forest for working with Centrify objects.
<code>Command : Right</code>	Represents the right to run a command, including which users and groups have that right.
<code>Commands</code>	Represents a collection of command rights.
<code>Computer</code>	Manages an individual computer account object.
<code>ComputerGroupUnixProfiles : GroupUnixProfiles</code>	Represents the groups in a computer zone.
<code>ComputerRole</code>	Manages a computer role.
<code>ComputerRoles</code>	Represents a collection of computer roles.
<code>Computers</code>	Represents a collection of computers in a zone.
<code>ComputerUserUnixProfiles : UserUnixProfiles</code>	Represents the users in a computer zone.
<code>Group</code>	Manages an individual group account object.
<code>GroupUnixProfile</code>	Manages the properties in the UNIX profile of a group.
<code>GroupUnixProfiles</code>	Represents a collection of UNIX groups in a zone.
<code>HierarchicalGroup : GroupUnixProfile</code>	Manages the properties in the UNIX profile of a group in a hierarchical zone.
<code>HierarchicalUser : UserUnixProfile</code>	Manages the properties in the UNIX profile of a user in a hierarchical zone.
<code>HierarchicalZone : Zone</code>	Represents a hierarchical zone.
<code>HierarchicalZoneComputer : Computer</code>	Manages the properties in the profile of a computer object joined to a hierarchical zone.
<code>HZRoleAssignment : RoleAssignment</code>	Manages a zone-level role assignment in a hierarchical zone.
<code>InheritedRoleAsg</code>	Represents an inherited role assignment.
<code>Key</code>	Represents a license key.
<code>Keys</code>	Represents a collection of Centrify license keys.
<code>License</code>	Represents a Centrify license.
<code>Licenses</code>	Represents a collection of Centrify licenses in a license container object.
<code>LicensesCollection</code>	Manages all the Centrify licenses in all of the Licenses parent containers defined for a forest.
<code>MZRoleAssignment : RoleAssignment</code>	Represents a computer-level role assignment.
<code>NetworkAccesses</code>	Represents a collection of network access rights.
<code>Pam : Right</code>	Represents a PAM (Pluggable Authentication Module)

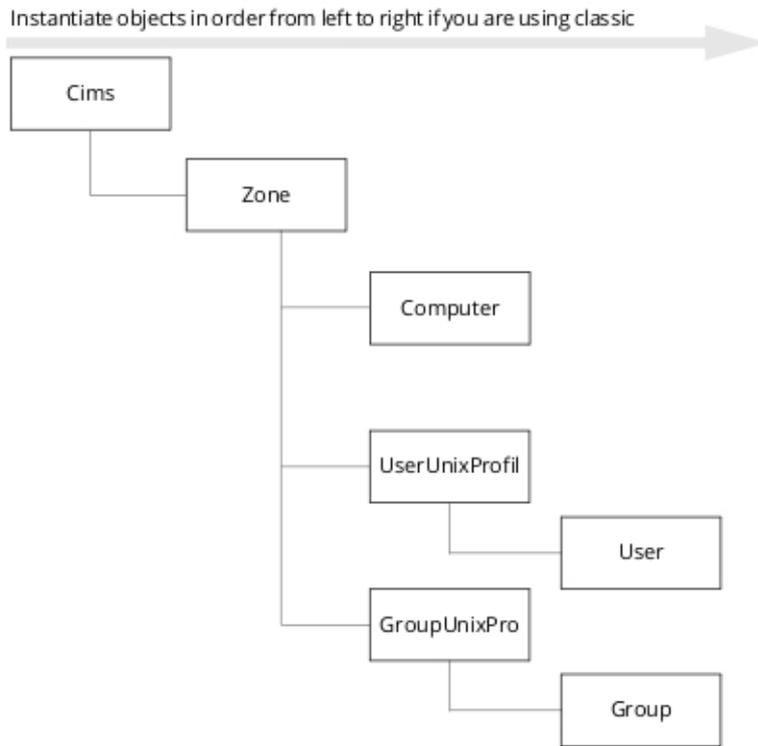


Class	Description
	application right.
Pams	Represents a collection of PAM application rights.
Right	The base class for all rights.
Right : NetworkAccess	Represents a Windows network access right.
Right : windowsApplication	Represents a Windows application right.
Right : windowsDesktop	Represents a Windows desktop right.
Role	Manages a Centrify role.
RoleAssignment	Represents a role assignment.
RoleAssignments	Represents a collection of role assignments.
Roles	Represents a collection of roles.
User	Represents an individual user account object.
UserUnixProfile	Manages the properties in the profile associated with an individual UNIX user.
UserUnixProfiles	Represents a collection of users in a zone.
windowsApplications	Represents a collection of Windows application rights.
windowsDesktops	Represents a collection of Windows desktop rights.
windowsUser	Represents a Windows user.
windowsUsers	Represents a collection of Windows users.
Zone	Represents a Centrify zone, including the users, groups, and computers that have been added to the zone.

In addition to these objects, there are optional objects for managing and manipulating NIS maps and NIS map entries in Active Directory. For an overview of those objects, see [Working with NIS maps](#). For more information about all of the objects that enable you to manipulate Centrify-specific data in Active Directory, see [Centrify object reference](#)

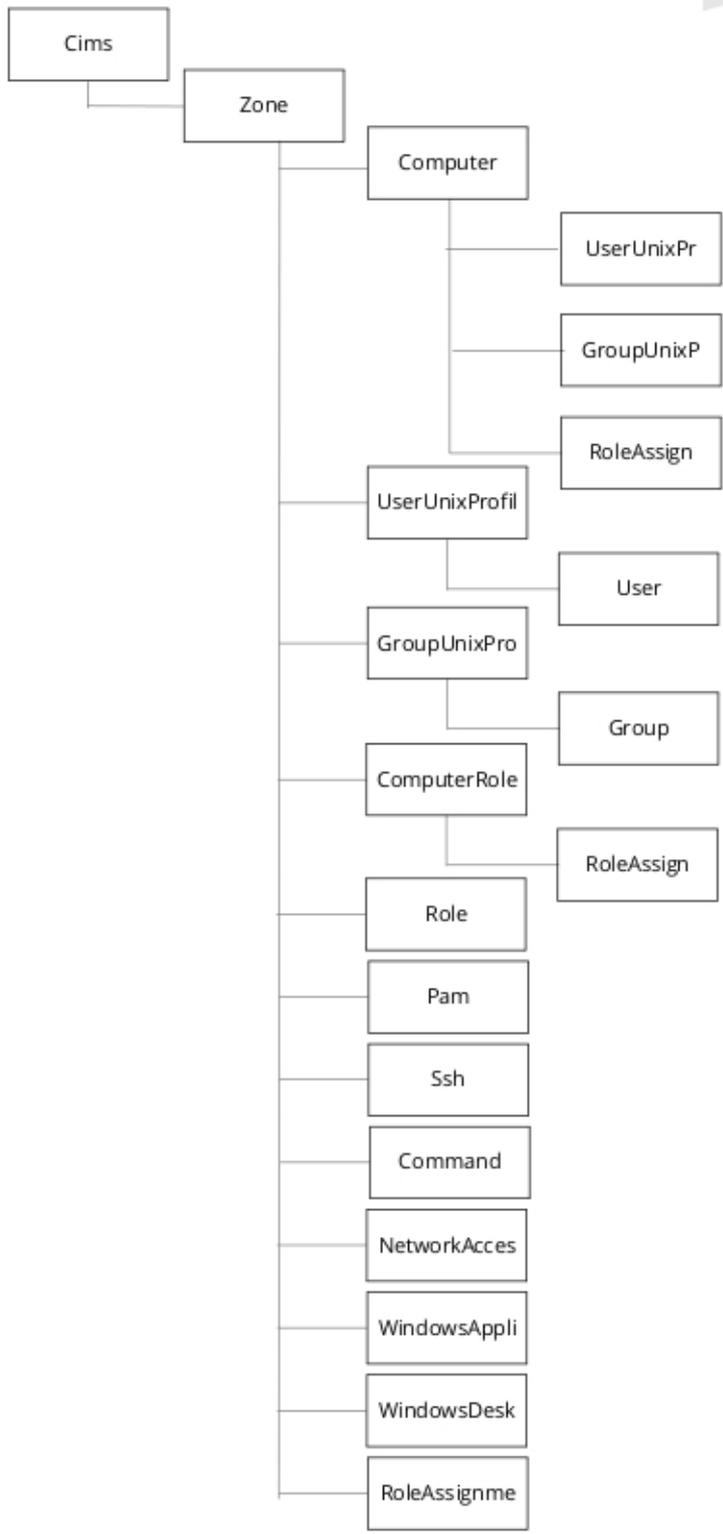
## Creating objects in the proper order

Centrify objects must be created in a particular order because some objects rely on the existence of others. For example, your application must create the `Cims` object first to establish communication with the Active Directory domain controller. After you create an instance of the `Cims` object, you must create the `Zone` object before you can create `User`, `Group`, or `Computer` objects because these objects exist in Active Directory in the context of the zone. The following figures illustrate the order for creating Centrify-related objects:





Instantiate objects in order from left to right if you are using 





## Getting and setting object properties

You can read or write most object properties; however a few are read-only. The syntax line in the object reference indicates whether an object property's value can be read (`{get;}`) or both read and written (`{get; set;}`). For example, the `nextAvailableUID` property can be both read and written:

```
int nextAvailableUID {get; set;}
```

To retrieve the existing value for this property, you could include a line similar to this:

```
read_uid_value = zone.nextAvailableUID
```

To set a new value for this property, you could include a line similar to this:

```
zone.nextAvailableUID = set_uid_value
```

## Interface naming conventions

The Centrify Windows API objects are stored in Active Directory using the IADs interface. The IADs interface is a Microsoft standard that defines basic object features—such as properties and methods—of any Active Directory Service Interface (ADSI) object. The most common ADSI objects include users, computers, services, file systems, and file service operations. The IADs interface ensures that all ADSI objects provide a simple and consistent representation of underlying directory services.

In addition to the basic ADSI objects, Centrify-specific objects are implemented as IADs interfaces. Using interfaces for the Centrify objects enables them to change internally without requiring any changes to the API. By convention, when objects are implemented as interfaces rather than class objects, they are identified by a capital “I” as a prefix. The Centrify-specific objects that are implemented as interface objects have the same names as the classes in **Centrify-specific objects classes**, with the addition of the “I” prefix; for example, the `IZone` interface object corresponds to the `Zone` class.

For more information about the IADs interface and working with interface objects, see the Microsoft Developer Network Library.



## Creating the top-level Cims object

The `Cims` object is the top-level object in the Centrify Windows API. This object is used to establish the connection with Active Directory and set up the environment so that other operations can be performed. Before you can retrieve any information from Active Directory, such as a zone object or user profile, you must create a `Cims` object. If you are writing COM-based for Centrify software, version 5.0 or later, the top-level `Cims` object is named `Cims3`. For example:

```
set cdc = CreateObject("Centrify.DirectControl.Cims3")
```

If you have scripts created for a previous version of Centrify software, you should modify the object created to be a `Cims3` object.

If you are writing programs using a .NET language, the namespace for the top-level `Cims` object is `Centrify.DirectControl.API.Cims`, regardless of the version of Centrify software you are using. For example, to create the top-level `Cims` object in a .NET program, you would type:

```
Centrify.DirectControl.API.Cims cdc = new  
Centrify.DirectControl.API.Cims();
```

After creating the top-level `Cims` object, you can use the other Centrify objects to access and manage zones, computers, user UNIX profiles, and group UNIX profiles (see [Creating objects in the proper order](#)). By writing scripts that retrieve or set object properties, you can provision users programmatically without using the Access Manager console or other MMC snap-ins.

## Working with NIS maps

In addition to the zone, computer, user, and group objects, you can use the Centrify Windows API to manage Network Information Service (NIS) maps in Active Directory. NIS maps store network information that can be used to respond to client requests on computers where the Centrify agent cannot be installed. You can create or import NIS maps using the Access Manager console or programmatically using the API. The NIS maps you create or import are zone-specific information in Active Directory. Once the information is stored in Active Directory, NIS clients can send requests to the Centrify Network Information Service (`adnisd`) to receive the data.

For more detailed information about working with NIS maps and the Centrify Network Information Service, see the *Planning and Deployment Guide* and the *NIS Administrator's Guide*.



The Centrify Windows API for working with NIS maps includes the following classes:

Class	Description
Store	Attaches to a zone and creates NIS maps in the zone.
Map	Works with an individual map and its records.
Entry	Manages the fields in an individual record.

## Writing scripts that use Centrify Windows API calls

To handle Centrify tasks programmatically, you can write programs that call Centrify Windows API functions using any of the tools commonly used to write programs for Windows-based operating environments. Some of the most common of these tools include VBScript, PowerShell, and Visual Studio (C#).

To illustrate using these tools, the following sections describe how to create and run a program that uses the Centrify objects to open a zone and lists all the users in it using VBScript and PowerShell. For more detailed examples of performing common tasks using these scripting languages, see the sample scripts included in the SDK package.

- [Using VBScript](#)
- [Using PowerShell](#)
- [Using Visual Studio C#](#)

### Using VBScript

In most cases, you can use VBScript to write scripts that call the Centrify Windows API.

The following steps illustrate how to create and run a VBScript script that uses the Centrify Windows API. This sample script opens a zone and lists all the users in it.

1. Verify that the computer you are using has Access Manager console or the Centrify Windows API Runtime environment from the Centrify SDK installed.
2. Verify that the computer you are using is a member of the Active Directory domain you want to work with.



3. Log in as a domain user with permission to read the zone data for the zone you will be listing.

If you can list the users in the zone using the Access Manager console with the credentials provided, you have the correct permissions. For information about configuring a user's rights to read zone data, see the *Planning and Deployment Guide*.

4. Use a text editor to create a file called `zone-list.vbs`.
5. Add the following text to `zone-list.vbs`, replacing the `domain_name` and the path to the zone with a domain name and zone location appropriate for your environment.

```
set cims = CreateObject("Centrify.DirectControl.Cims3")
set zone = cims.GetZone("domain_name/zone_path/zone_name")
set users = zone.GetUserUnixProfiles()
```

```
for each user in users
  if (user.IsNameDefined) then
    name = user.Name
  else
    name = "<Empty>"
  end if
```

```
  if (user.IsUidDefined) then
    uid = user.Uid
  else
    uid = "<Empty>"
  end if
```

```
  wscript.echo name & " | " & uid
next
```

For example if you are using the domain `test.acme.com` and want to list users in the "default" zone in its default container location:

```
set zone = cims.getzone("test.acme.com/program
data/centrify/zones/default")
for each user in users
  wscript.echo user.name, user.Uid
next
```

6. Click **Start > Run**, then type `cmd` to open a command window.
7. Change directory to the location of the VBScript file and type:  
`cscript zone-list.vbs`

You should see output similar to the following:

```
C:\>cscript zone-list.vbs
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights
reserved.
```

```
jane 10000
```



```
jim.smit 10002  
jimsmith 10003  
joe 10004  
paul 10006  
rachel 10016
```

## Using PowerShell

Centrify provides a separate Access Module for PowerShell that includes predefined “cmdlets” for performing a broad range of administrative tasks without requiring any knowledge of the underlying API calls. If you prefer, however, you can write PowerShell scripts that call the Centrify Windows API directly. The following steps illustrate how to create and run a sample script that opens a zone and lists all the users in it.

1. Verify that the computer you are using has Access Manager or the Centrify Windows API Runtime environment from the Centrify SDK installed.
2. Verify that the computer you are using is a member of the Active Directory domain you want to work with.
3. Log in as a domain user with permission to read the zone data for the zone you will be listing.

If you can list the users in the zone using Access Manager with the credentials provided, you have the correct permissions. For information about configuring a user’s rights to read zone data, see the *Planning and Deployment Guide*.

4. Use a text editor to open the sample script file `util.ps1`.
5. Modify the `util.ps1` script to specify a user name and password with administrative access to the Active Directory domain.

For example, replace the “\*\*\*\*\*” string with an administrator user name and password:

```
$username = "administrator";  
$passwd = "password";
```

6. Use a text editor to create a file called `zone-list.ps1`.
7. Add the following text to `zone-list.ps1`, replacing the `domain_name` and the path to the zone with a domain controller and zone location appropriate for your environment.

```
$api = "Centrify.DirectControl.API.{0}";  
$cims = New-Object($api -f "Cims");  
$objZone = $cims.GetZone("domain_name/zone_path/zone_name");
```



```
$users = $objZone.GetUserUnixProfiles();  
foreach ($user in $users)  
{  
    if ($objZone.IsHierarchical)  
    {  
        if ($user.IsNameDefined)  
        {  
            $name = $user.Name;  
        }  
        else  
        {  
            $name = "<Empty>";  
        }  
        if ($user.IsUidDefined)  
        {  
            $uid = $user.UID;  
        }  
        else  
        {  
            $uid = "<Empty>";  
        }  
    }  
    else  
    {  
        $name = $user.Name;  
        $uid = $user.UID;  
    }  
    write-Host ("{0} | {1}" -f $name, $uid);  
}
```

For example if you are using the domain `test.acme.com` and want to list users in the “global” zone in its default container location:

```
var zone = cims.getzone("test.acme.com/program  
data/centrify/zones/global");
```

8. Click **Start > Run**, then type `cmd` to open a command window.
9. Change directory to the location of the script file and type the following to run the script using Windows Script Host:

```
cscript zone-list.ps1
```

You should see output similar to the output for the VBScript sample script. For information about using the Access Module for PowerShell instead of writing scripts that call the Centrify Windows API, see the *Access Control and Privilege Management Scripting Guide*.



## Using Visual Studio C#

The following steps describe how to call the Centrify Windows API when using Visual Studio 2010. Alternatively you can use the command line compilers that come in Microsoft .Net Framework SDK or the Visual Studio Express Edition. The example below is created using C#, however using **vb.net** is very similar.

Note that the .NET assemblies are not installed in the Global Assembly Cache, but they do have version numbers on them. This means that the calling applications are tied to using the same assembly versions they were compiled with. To avoid problems using the assemblies, you should install the assemblies and the applications that use the assemblies in the same directory.

1. Verify that the computer you are using has Access Manager or the Centrify Windows API Runtime environment from the Centrify SDK installed.
2. Verify that the computer you are using is a member of the Active Directory domain you want to work with.
3. Log in as a domain user with permission to read the zone data for the zone you will be listing.

If you can list the users in the zone using Access Manager with the credentials provided, you have the correct permissions. For information about configuring a user's rights to read zone data, see the *Planning and Deployment Guide*.

4. Start **vs2010** and start a new project of type **C# console application**.
5. Click **Project > Add reference**.
6. Click the **.NET** tab, then click **Browse**.
7. Navigate to the directory where Access Manager or the SDK is installed. For example, browse to the default location `C:\Program Files\Centrify\`.
8. Select the following dynamic link libraries to add:

```
centrifydc.api.dll  
interface.dll  
nismap.api.dll  
PropSheetHost.dll  
util.dll
```

9. Add a reference to **system.directory** services. From the **Project** menu, select **Add references**. In the **.NET** tab scroll down to `system.directoryservices.dll`.



10. Open the class file that contains the application's Main function. By default, Visual Studio creates this file as `class1.cs`.
11. Add the following code in the **Main** function, replacing the `domain_name` and the path to the zone with a domain controller and zone location appropriate for your environment:

```
Centrify.DirectControl.API.Cims cims = new
Centrify.DirectControl.API.Cims();
Centrify.DirectControl.API.IZone zone =
cims.GetZone("domain_name/zone_path/zone_name");
foreach (Centrify.DirectControl.API.IUserUnixProfile user
in zone.GetUserUnixProfiles())
{
    string name, uid;
    if (zone.IsHierarchical &&
!
((Centrify.DirectControl.API.CDC50.UserUnixProfile)user).IsNameDefined)
    {
        name = "<Empty>";
    }
    else
    {
        name = user.Name;
    }
    if (zone.IsHierarchical &&
!
((Centrify.DirectControl.API.CDC50.UserUnixProfile)user).IsUIDDefined)
    {
        uid = "<Empty>";
    }
    else
    {
        uid = user.UID.ToString();
    }
    Console.WriteLine(name + " | " + uid);
}
```

For example if you are using the domain `dc2k.seattle.test` and want to list users in the "default" zone in its default container location:

```
Centrify.DirectControl.API.IZone zone =
cims.GetZone("dc2k.seattle.test/program
data/centrify/zones/default");
```

12. Press **F5** to compile and run the application.



# Centrify object reference

This chapter describes the classes, methods, and properties available for working with Centrify objects for access control and privilege management. The primary classes for working with Centrify data objects are defined in the `Centrify.DirectControl.API` namespace and consist of:

- `AzRoleAssignment`
- `Cims`
- `Command`
- `Commands`
- `Computer`
- `ComputerGroupUnixProfiles`
- `ComputerRole`
- `ComputerRoles`
- `Computers`
- `ComputerUserUnixProfiles`
- `CustomAttributeContainer`
- `CustomAttributes`
- `CustomAttribute`
- `Group`
- `GroupUnixProfile`
- `GroupUnixProfiles`
- `HierarchicalGroup`
- `HierarchicalUser`
- `HierarchicalZone`
- `HierarchicalZoneComputer`



- HzRoleAssignment
- InheritedRoleAsg
- Key
- Keys
- License
- Licenses
- LicensesCollection
- MzRoleAssignment
- NetworkAccess
- NetworkAccesses
- Pam
- Pams
- Right
- Role
- RoleAssignment
- RoleAssignments
- Roles
- Ssh
- Sshs
- User
- UserUnixProfile
- UserUnixProfiles
- WindowsApplication
- WindowsApplicationCriteria
- WindowsApplications
- WindowsDesktop
- WindowsDesktops
- WindowsUser
- WindowsUsers



- **Zone**

In addition to the basic classes, the following classes are defined in the `Centrify.DirectControl.NISMap.API` namespace for working with NIS maps:

- **Entry**
- **Map**
- **Store**

There are also separate classes for pending import groups and users. The following classes are defined in the `Centrify.DirectControl.API.Import` namespace for working with groups and users imported from UNIX with the “Import from UNIX” wizard:

- **GroupInfo**
- **GroupInfos**
- **GroupMember**
- **GroupMembers**
- **UserInfo**
- **UserInfos**

## AzRoleAssignment

The `AzRoleAssignment` class represents a computer role assignment, where a role assignment object contains information about an Active Directory object (trustee—that is, user or group) that has been added to a computer role.

### Syntax

```
public interface IAzRoleAssignment : IRoleAssignment
```

### Methods

The `AzRoleAssignment` class provides the following methods:



This method	Does this
<code>Commit</code>	Commits changes in the role to Active Directory. (Inherited from <code>RoleAssignment</code> .)
<code>ClearCustomAttributes</code>	VBScript interface to clear the custom attributes for this class. (Inherited from <code>RoleAssignment</code> .)
<code>Delete</code>	Deletes the role. (Inherited from <code>RoleAssignment</code> .)
<code>GetComputerRole</code>	Returns the computer role that logically contains this role assignment. (Inherited from <code>RoleAssignment</code> .)
<code>GetTrustee</code>	Returns the trustee being assigned. (Inherited from <code>RoleAssignment</code> .)
<code>ICustomAttributeContainer</code> <code>GetCustomAttributeContainer</code>	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class. (Inherited from <code>RoleAssignment</code> .)
<code>SetCustomAttribute</code>	VBScript interface to set the custom attributes for this class. (Inherited from <code>RoleAssignment</code> .)
<code>validate</code>	Validates this role assignment. (Inherited from <code>RoleAssignment</code> .)

## Properties

The `AzRoleAssignment` class provides the following properties:

This property	Does this
<code>CustomAttributes</code>	VBScript only: Gets or sets custom attributes for this class. (Inherited from <code>RoleAssignment</code> .)
<code>EndTime</code>	Determines the time at which this role becomes inactive. (Inherited from <code>RoleAssignment</code> .)
<code>Id</code>	Gets the GUID of the role assignment. (Inherited from <code>RoleAssignment</code> .)
<code>IsRoleOrphaned</code>	Indicates whether the role assignment is orphaned due to missing or invalid data.



This property	Does this
	(Inherited from <a href="#">RoleAssignment</a> .)
<a href="#">IsTrusteeOrphaned</a>	Indicates whether the role assignment is orphaned due to a missing trustee. (Inherited from <a href="#">RoleAssignment</a> .)
<a href="#">LocalTrustee</a>	Gets the local trustee being assigned. (Inherited from <a href="#">RoleAssignment</a> .)
<a href="#">Role</a>	Gets the role the trustee is assigned to. (Inherited from <a href="#">RoleAssignment</a> .)
<a href="#">StartTime</a>	Specifies the time from which this role becomes effective. (Inherited from <a href="#">RoleAssignment</a> .)
<a href="#">TrusteeDn</a>	Gets the distinguished name of the trustee assigned the role. (Inherited from <a href="#">RoleAssignment</a> .)
<a href="#">TrusteeType</a>	Gets the trustee type of the role assignment. (Inherited from <a href="#">RoleAssignment</a> .)

## Discussion

A computer role describes the intended use of a group of computers; for example, the set of computers dedicated as database servers. See [ComputerRoles](#) for a discussion of computer roles.

## GetComputerRole

Gets the computer role that logically contains this role assignment.

### Syntax

```
IComputerRole GetComputerRole()
```

### Return value

The `ComputerRole` instance containing this role assignment.



## Discussion

The role assignment contains information about an Active Directory object that has been assigned to a computer role.

## Exceptions

`GetComputerRole` throws an `ApplicationException` if no computer role is found, multiple computer roles are found, or an error occurs when accessing Active Directory.

## Example

The following code sample illustrates using this method in a script:

```
...  
IComputerRole compRole = objZone.GetComputerRole(strName);  
if (compRole != null)  
{  
    Console.WriteLine("Computer role " + strName + " already  
exists.");  
}  
...
```

## Cims

The `Cims` class is the top-level class in the Centrify Windows API.

## Syntax

```
public interface ICims
```

## Discussion

This class is used to establish the connection with Active Directory and set up the environment so that other operations can be performed. Before you can retrieve any information from Active Directory, you must create a `Cims` object. For example:

```
'Create a CIMS object to interact with Active Directory  
set cims = CreateObject("Centrify.DirectControl.Cims")
```



If you are writing programs using Centrify, version 5.0 or later, the top-level Cims object is named Cims3. For example:

```
set cdc = CreateObject("Centrify.DirectControl.Cims3")
```

If you have scripts created for a previous version of Centrify software, you should modify the object created to be a Cims3 object to work with version 5.0 or later.

If you are writing programs using a .NET language, the namespace for the top-level Cims object is Centrify.DirectControl.API.Cims, regardless of the version of Centrify you are using. For example, to create the top-level Cims object in a .NET program, type:

```
Centrify.DirectControl.API.Cims cdc = new  
Centrify.DirectControl.API.Cims();
```

## Methods

The Cims class provides the following methods:

This method	Does this
AddComputer	Adds a computer object to a specific zone.
AddComputerZone	Adds a computer zone to a computer object.
AddWindowsComputer	Adds a Windows computer object to a hierarchical zone.
ConfigureForest	Configures the Active Directory forest to work with Centrify software.
Connect	Connects to an Active Directory domain controller.
CreateZone	Creates an individual zone object in a parent container object.
CreateZoneWithSchema	Creates an individual zone object with a specified schema type in a parent container object.
GetComputer	Returns a computer object with its related data by its directory object.
GetComputerByComputerZone	Returns a computer object given the LDAP path to the computer zone.
GetComputerByPath	Returns a computer object given the LDAP path to the computer.
GetGroup	Returns a group object with its related data by its directory object.
GetGroupByPath	Returns a group object with its related data by its LDAP path.
GetUser	Returns a user object with its related data by its directory



This method	Does this
	object.
<code>GetUserByPath</code>	Returns a user object with its related data by its LDAP path.
<code>GetWindowsUser</code>	Returns a Windows user object.
<code>GetWindowsUserByPath</code>	Returns a Windows user object given the path to the object.
<code>GetZone</code>	Returns a zone object with its related data by object name.
<code>GetZoneByPath</code>	Returns a zone object with its related data by its LDAP path.
<code>IsForestConfigured</code>	Checks whether the forest is properly configured with valid Centrify licenses.
<code>LoadLicenses</code>	Returns all of the Centrify licenses for the connected domain.

## Properties

The `Cims` class provides the following properties:

This property	Does this
<code>Password</code>	Gets the password used to establish the connection to the Active Directory domain.
<code>Server</code>	Gets the domain controller computer name used to establish the connection to the Active Directory domain.
<code>UserName</code>	Gets the user name used to establish the connection to the Active Directory domain.

## AddComputer

Adds a computer object to a specific zone.

### Syntax

```
IComputer AddComputer(IADs computer, IZone zone)
```

```
IComputer AddComputer(string computerDn, IZone zone)
```

### Parameters

Specify one of the following parameters when using this method.



Parameter	Description
computer	The Active Directory computer object that you wish to add to the zone.
computerDn	The distinguished name of the computer object.

Specify the following parameter when using this method.

Parameter	Description
zone	The zone to which you wish to add the computer object.

## Return value

The newly-added computer object.

## Exceptions

AddComputer throws an `ArgumentNullException` if any parameter value is null or empty.

## AddComputerZone

Adds a computer zone to a computer object.

## Syntax

```
IHierarchicalZoneComputer AddComputerZone(string dnsname, IZone zone)
```

## Parameters

Specify the following parameters when using this method.

Parameter	Description
dnsname	The DNS host name of the Active Directory computer object to which you wish to add a computer zone.
zone	The hierarchical zone to which the computer object belongs.

## Return value

The hierarchical computer object that contains the computer zone.



## Discussion

Computer-level overrides for user, group, or computer role assignments are contained in a computer zone, a Centrify zone in Active Directory that contains properties that are specific to only one computer. Computer zones are not exposed in Access Manager.

This method adds a computer zone to a computer object in a hierarchical zone. If the Active Directory computer object exists, the method adds the computer zone to that computer. If the computer object does not exist, the method creates an orphan computer zone. When you create an Active Directory computer with the same DNS host name and call the **AddComputer** method to add it to the zone, this computer zone is linked to that computer object.

## Exceptions

AddComputerZone may throw one of the following exceptions:

- `ArgumentNullException` if the DNS name parameter value is `null`.
- `ArgumentException` if the DNS name is not valid or the zone is not recognized.

## AddWindowsComputer

Adds a Windows computer object to a hierarchical zone.

## Syntax

```
IComputer AddwindowsComputer(IADS computer, IHierarchicalZone zone)
```

```
IComputer AddwindowsComputer(string computerDn, IHierarchicalZone zone)
```

## Parameters

Specify one of the following parameters when using this method.

Parameter	Description
<code>computer</code>	The Active Directory computer object that you wish to add to the zone.
<code>computerDn</code>	The distinguished name of the computer object.

Specify the following parameter when using this method.



---

Parameter	Description
zone	The zone to which you wish to add the computer object.

---

## Return value

The newly-added computer object.

## Exceptions

`AddWindowsComputer` throws an `ArgumentNullException` if any parameter value is `null` or empty.

## ConfigureForest

Configures the Active Directory forest to work with Centrify software.

## Syntax

```
void ConfigureForest(string licenseContainerPath)
```

## Parameters

Specify the following parameter when using this method.

---

Parameter	Description
<code>licenseContainerPath</code>	The LDAP path to the license container holding your Centrify licenses.

---

## Discussion

Your Centrify license container must be set up before calling this function. See the `Licenses` class for more information about license containers.

## Exceptions

`ConfigureForest` throws an `ArgumentException` if the parameter value is `null` or empty or if the method cannot find the license container object.

## Connect

Establishes a connection to an Active Directory domain controller.



## Syntax

```
void Connect(string server, string username, string password)
```

## Parameters

Specify the following parameters when using this method.

Parameter	Description
server	The name of the Active Directory domain controller to which you are establishing a connection.
username	The Active Directory user account for connecting to the domain controller. The rights associated with this account used to establish the connection to Active Directory can control the operations you are allowed to perform in a script.
password	The password for the Active Directory user account connecting to the domain controller.

## Discussion

This method enables you to connect to a specific domain controller using a specific user name and password, if the Active Directory server name, user name, and user password are all valid. This method is not required when you connect to Active Directory using the credentials you used to log on to the computer.

Call `Cims.Connect("domaincontroller", NULL, NULL)` to use the default user account

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify credentials to use for connecting to Active Directory  
cims.Connect("ginger.ajax.org", "dane", "Niles9!");  
...
```

## CreateZone

Creates a zone in the specified parent container and returns the zone object created.

## Syntax

```
IZone CreateZone(IADS container, string name)
```



## Parameters

Specify the following parameters when using this method.

Parameter	Description
<code>container</code>	The IADs interface of the parent container object to be used to store the new zone. You can use the standard ADSI <code>GetObject</code> function to retrieve this interface.
<code>name</code>	The name of the new zone.

## Return value

The zone object and its related data as `Centrify.DirectoryControl.API.IZone`.

## Discussion

The `CreateZone` function requires you to specify the Active Directory container object or organizational unit where the zone should be created. You can use the Active Directory `GetObject` function to retrieve the ADSI pointer to the specified container.

## Exceptions

`CreateZone` may throw one of the following exceptions:

- `ArgumentNullException` if the container object is a `null` reference.
- `ArgumentException` if the zone name is invalid.
- `ApplicationException` if a global catalog server error occurs.
- `UnauthorizedAccessException` if the container object cannot be read because of insufficient permissions.
- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using this method in a script to create a new hierarchical zone named `sample_zone` in the parent container `Program Data/Centrify/Zones`:

```
...  
string strParent = "CN=zones,CN=Centrify,CN=Program Data";
```



```

string strZone = "sample_zone";
// Create a CIMS object to interact with AD.
Cims cims = new Cims();
// Note the lack of the cims.connect function.
// By default, this script will use the connection to the domain
controller
// and existing credentials from the computer already logged in.
// Obtain an active directory container object.
DirectoryEntry objRootDSE = new DirectoryEntry("LDAP://rootDSE");
DirectoryEntry objContainer = new DirectoryEntry("LDAP://" +
strParent + "," + objRootDSE.Properties
["defaultNamingContext"].Value.ToString());
IHierarchicalZone objZone = cims.CreateZone(objContainer,
strZone) as IHierarchicalZone;
...

```

## CreateZoneWithSchema

Creates a zone with a specified schema type in the specified parent container and returns the zone object created.

### Syntax

```

IZone CreateZoneWithSchema(IADs container, string name, int
schema, int objectType)

```

### Parameters

Specify the following parameters when using this method.

Parameter	Description
container	The IADs interface of the parent container object to be used to store the new zone.
name	The name of the new zone.
schema	The schema type to use for the new zone. This parameter determines how the zone data is stored in Active Directory. For more information about the valid schema types you can specify, see <a href="#">Schema</a> .
objectType	The Active Directory object type to use for the zone. The valid values are: <ul style="list-style-type: none"> <li>■ 0 defines the zone object as a <b>Container</b> object.</li> <li>■ 1 defines the zone object as an <b>Organization Unit</b>.</li> </ul>

### Return value

The zone object as `Centrifify.DirectControl.API.IZone`.



## Discussion

The `CreateZoneWithSchema` function requires you to specify the Active Directory container object or organizational unit where the zone should be created. You can use the standard Active Directory `GetObject` function to retrieve the ADSI pointer to the specified container.

## Exceptions

`CreateZoneWithSchema` may throw one of the following exceptions:

- `ArgumentNullException` if the container object is a `null` reference.
- `ArgumentException` if the zone name is invalid.
- `ApplicationException` if a global catalog server error occurs.
- `UnauthorizedAccessException` if the container object cannot be read because of insufficient permissions.
- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using this method in a script to create a new classic zone named `ConsumerDiv` as an organization unit in the parent container `ajax.org/Corporate/Zones`:

```
...  
'Specify the parent container location for the zone  
set objContainer = GetObject("LDAP://cn=Zones,cn=Corporate,  
dc=ajax,dc=org")  
'Create a new zone named "ConsumerDiv"  
set objZone = cims.CreateZoneWithSchema(objContainer,  
"ConsumerDiv", 3, 1)  
...
```

The `GetObject` call retrieves the ADSI pointer to the specified container.

## GetComputer

Returns a computer object with all of its related Centrify-specific data, including all of the Computer object's properties and methods.



## Syntax

`IComputer GetComputer(IADS computer)`

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>computer</code>	The IADs interface to the computer object you want to retrieve. You can use the standard ADSI <code>GetObject</code> function to retrieve this interface.

## Return value

The computer object as:

`Centrify.DirectControl.API.IComputer`

## Discussion

This method returns the computer object using the IADs interface to locate the object. The IADs interface is the directory object that represents the computer in Active Directory. The IADs object is useful for retrieving Active Directory-specific information, such as the site, for a computer object.

The method returns the computer object as `Centrify.DirectControl.API.IComputer`. You can then use the `IComputer` object to retrieve Centrify-specific information, such as the version of the Centrify agent installed on the computer.

## Exceptions

`GetComputer` throws an `ArgumentException` if the computer path is null or empty.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/east_div")  
'Identify the computer you want to work with  
set objIADsComputer = GetObject("LDAP://CN=magnolia,  
CN=Computers,DC=ajax,DC=org")  
'Get the directory object for the computer  
set objComputer = cims.GetComputer(objIADsComputer)  
...
```



## GetComputerByComputerZone

Returns a computer object with all of its related Centrify-specific data, given the path to the computer zone associated with the computer.

### Syntax

```
IHierarchicalZoneComputer GetComputerByComputerZone(string zonepath)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
zonepath	The LDAP path or distinguished name of the computer zone of the computer object you want to retrieve.

### Return value

The computer object as:

```
Centrify.DirectControl.API.IHierarchicalZoneComputer
```

### Discussion

When you assign computer-level overrides for user, group, or computer role assignments, the Centrify Windows API creates a computer zone, a Centrify zone in Active Directory that contains the users, groups, and computer role assignments that are specific to only that one computer. Computer zones are not exposed in Access Manager.

This method returns the computer object using the LDAP path or distinguished name of the computer zone. The LDAP path to a computer zone uses the following format:

```
LDAP://[domain/]attr=name,[...],dc=domain_part,[...]
```

For example, if you use the default parent location for computer accounts in the domain `arcade.com`, the LDAP path for the computer account `magnolia` is:

```
LDAP://cn=magnolia,cn=Computers,dc=arcade,dc=com
```



## Exceptions

`GetComputerByComputerZone` throws an `ApplicationException` if the method cannot find the specified computer.

## GetComputerByPath

Returns a computer object with all of its related Centrify-specific data, including all of the Computer object's properties and methods.

## Syntax

```
IComputer GetComputerByPath(string path)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>path</code>	The LDAP path or distinguished name of the computer object you want to retrieve.

## Return value

The computer object as:

```
Centrify.DirectControl.API.IComputer
```

## Discussion

This method returns the computer object using the LDAP path or distinguished name of the object. The LDAP path to a computer account uses the following format:

```
LDAP://[domain/]attr=name,[...],dc=domain_part,[...]
```

For example, if you use the default parent location for computer accounts in the domain `arcade.com`, the LDAP path for the computer account `magnolia` is:

```
LDAP://cn=magnolia,cn=Computers,dc=arcade,dc=com
```

The method returns the computer object as `Centrify.DirectControl.API.IComputer`.



## Exceptions

GetComputerByPath throws an ArgumentException if the computer path is null or empty.

## Example

The following code sample illustrates using this method in a script:

```
...
'Specify the zone you want to work with
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/east_div")
'Identify the computer you want to work with
Set objComputer = cims.GetComputerByPath("LDAP://cn=magnolia,
cn=computers,dc=ajax,dc=org")
...
```

## GetGroup

Returns an Active Directory group object with all of its related Centrify-specific data.

## Syntax

```
IGroup GetGroup(IADs group)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
group	The IADs interface to the group object you want to retrieve. You can use the standard ADSI GetObject function to retrieve this interface.

## Return value

The group object as:

```
Centrify.DirectControl.API.IGroup
```

## Discussion

This method uses the IADs interface to locate the group object. The IADs interface is the directory object that represents the group in Active Directory.



The IADs object is useful for retrieving Active Directory-specific information, such as the site, for a group.

The method returns the group object as `Centrify.DirectControl.API.IGroup`. You can then use the `IGroup` object to retrieve Centrify-specific information. For example, you can use `IGroup.unixProfiles` to retrieve the UNIX group profiles associated with an Active Directory group or `IGroup.AddunixProfile` to add a UNIX group profile for an Active Directory group to the zone.

## Exceptions

`GetGroup` throws an `ArgumentException` if the parameter is null or empty.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/east_div")  
'Identify the Active Directory group you want to work with  
set objIADsGroup = GetObject("LDAP://CN=IT Interns,CN=Users,  
DC=ajax,DC=org")  
'Get the directory object for the group  
set objGroup = cims.GetGroup(objIADsGroup)  
...
```

## GetGroupByPath

Returns an Active Directory group object with all of its related Centrify-specific data given the path to the object.

## Syntax

```
IGroup GetGroupByPath(string path)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
path	The LDAP path or distinguished name of the group object you want to retrieve.



## Return value

The group object as:

```
Centrify.DirectControl.API.IGroup
```

## Discussion

This method returns the group object using the LDAP path or distinguished name of the object. The LDAP path to a group uses the following format:

```
LDAP://[domain/]attr=name,[...],dc=domain_part,[...]
```

For example, if you use the default parent location for groups in the domain `arcade.com`, the LDAP path for the IT Interns group is:

```
LDAP://cn=IT Interns,cn=Users,dc=arcade,dc=com
```

The method returns the group object as

```
Centrify.DirectControl.API.Group.ObjectName.
```

## Exceptions

`GetGroupByPath` throws an `ArgumentException` if the group path is null or empty.

## Example

The following code sample illustrates using this method in a script:

```
...
string strParent = "CN=zones,CN=Centrify,CN=Program Data";
if (args.Length != 2)
{
    Console.WriteLine("Usage:");
    Console.WriteLine("    test_remove_group.exe
        \"cn=sample_group,ou=groups,dc=domain,dc=tld\"
        \"default\");
    return;
}
string strGroup = args[0];
string strZone = args[1];
//Need to obtain an active directory container object
DirectoryEntry objRootDSE = new DirectoryEntry("LDAP://rootDSE");
DirectoryEntry objContainer = new DirectoryEntry("LDAP://" +
strParent + "," + objRootDSE.Properties
["defaultNamingContext"].Value.ToString());
string strContainerDN = objContainer.Properties
["DistinguishedName"].Value as string;
//Create a CIMS object to interact with AD
ICims cims = new Cims();
//Note the lack of the cims.connect function.
//By default, this application will use the connection to the
```



```
domain controller
//and existing credentials from the computer already logged in.
//Get the group object
IGroup objGroup = cims.GetGroupByPath(strGroup);
//Get the zone object
IZone objZone = cims.GetZoneByPath("cn=" + strZone + "," +
strContainerDN);
...
```

## GetUser

Returns an Active Directory user object with all of its related Centrify-specific data.

### Syntax

```
IUser GetUser(IADs user)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
user	The IADs interface to the user object you want to retrieve.

### Return value

The user object as:

```
Centrify.DirectControl.API.IUser
```

### Discussion

This method uses the IADs interface to locate the user object. The IADs interface is the directory object that represents the user in Active Directory. The IADs object is useful for retrieving Active Directory-specific information, such as the site, for a user.

The method returns the user object as `Centrify.DirectControl.API.IUser`. You can then use the `IUser` object to retrieve Centrify-specific information.

### Exceptions

`GetUser` throws an `ArgumentException` if the parameter is `null` or empty.



## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/east_div")  
'Identify the Active Directory user you want to work with  
Set objUser = cims.GetUser("ajax.org/Users/Jae Smith")  
...
```

## GetUserByPath

Returns an Active Directory user object with all of its related Centrify-specific data given the path to the object.

### Syntax

```
IUser GetUserByPath(string path)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
path	The LDAP path or distinguished name of the user object you want to retrieve.

### Return value

The user object as:

```
Centrify.DirectControl.API.IUser
```

### Exceptions

`GetUserByPath` throws an `ArgumentException` if the computer path is null or empty.

### Discussion

This method returns the user object using the LDAP path or distinguished name of the object. The LDAP path to a user object uses the following format:

```
LDAP://[domain/]attr=name,[...],dc=domain_part,[...]
```



The method returns the user object as `Centrify.DirectControl.API.User.ObjectName`. For example, if the Active Directory user account is `Jae Smith` and the LDAP path to the account is `CN=Jae Smith, CN=Users, DC=ajax, DC=org`, the method returns the user object as:

```
Centrify.DirectControl.API.User.Jae Smith
```

## Example

The following code sample illustrates using this method in a script:

```
...
string strUser = args[0];
if (string.IsNullOrEmpty(strUser))
{
    Console.WriteLine("User DN cannot be empty.");
    return;
}
// Obtain an active directory container object
// Configure the test container
DirectoryEntry objRootDSE = new DirectoryEntry("LDAP://rootDSE");
DirectoryEntry objContainer = new DirectoryEntry("LDAP://" +
strParent + "," + objRootDSE.Properties
["defaultNamingContext"].Value.ToString());
string strContainerDN = objContainer.Properties
["DistinguishedName"].Value as string;

// Create a CIMS object to interact with AD
ICims cims = new Cims();

// Note the lack of the cims.connect function.'
// By default, this application will use the connection to domain
controller
// and existing credentials from the computer already logged in.

IHierarchicalZone objZone =
cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
IUser objUser = cims.GetUserByPath(strUser);
if (objUser == null)
{
    Console.WriteLine("User " + strUser + " does not exist.");
    return;
}
...
```

## GetWindowsUser

Returns a Windows user object.



## Syntax

```
IWindowsUser GetWindowsUser(IADs user)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
user	The IADs interface to the user object you want to retrieve.

## Return value

The user object as:

```
Centrify.DirectControl.API.IUser
```

## Exceptions

`GetWindowsUser` throws an `ArgumentException` if the parameter is `null` or empty.

## Discussion

This method uses the IADs interface to locate the user object. The IADs interface is the directory object that represents the user in Active Directory. The IADs object is useful for retrieving Active Directory-specific information, such as the site, for a user.

The method returns the user object as `Centrify.DirectControl.API.IUser`. You can then use the `IUser` object to retrieve Centrify-specific information.

## GetWindowsUserByPath

Returns a Windows user object given the path to the object.

## Syntax

```
IUser GetWindowsUserByPath(string path)
```

## Parameter

Specify the following parameter when using this method:



---

Parameter	Description
path	The LDAP path or distinguished name of the user object you want to retrieve.

---

## Return value

The user object as:

```
Centrify.DirectControl.API.IUser
```

## Exceptions

GetWindowsUserByPath throws an `ArgumentException` if the path is null or empty.

## Discussion

This method returns the user object using the LDAP path or distinguished name of the object. The LDAP path to a user object uses the following format:

```
LDAP://[domain/]attr=name,[...],dc=domain_part,[...]
```

The method returns the user object as

`Centrify.DirectControl.API.User.ObjectName`. For example, if the Active Directory user account is Jae Smith and the LDAP path to the account is `CN=Jae Smith, CN=Users, DC=ajax, DC=org`, the method returns the user object as:

```
Centrify.DirectControl.API.User.Jae Smith
```

## GetZone

Returns a zone object with all of its related Centrify-specific data given the zone name.

## Syntax

```
IZone GetZone(string zoneName)
```

## Parameter

Specify the following parameter when using this method:

---

Parameter	Description
zoneName	The name of the individual zone object to retrieve.

---



## Return value

If the operation is successful, `GetZone` returns the named zone object and its related data as:

`Centrify.DirectControl.API.IZone`

## Discussion

This method requires the full path to the individual zone object you want to retrieve.

This method uses the Active Directory canonical name for the zone. The canonical name for the zone uses the following naming structure:

`domain_name/container_name/[container_name...]/zone_name`

For example, if you use the default parent location for zones, the canonical name for the “default” zone is:

`domain_name/Program Data/Centrify/Zones/default`

## Exceptions

`GetZone` may throw one of the following exceptions:

- `ArgumentNullException` if no `zoneName` parameter is passed.
- `ApplicationException` if the specified zone name is not valid.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/east_div")  
...
```

## GetZoneByPath

Returns a zone object with all of its related Centrify-specific data given its LDAP path.

## Syntax

`IZone GetZoneByPath(string path)`



## Parameter

Specify the following parameter when using this method:

Parameter	Description
path	The full LDAP path to the individual zone object you want to retrieve.

## Return value

If the operation is successful, `GetZoneByPath` returns the zone object and its related data as `Centrify.DirectControl.API.IZone`.

## Discussion

The LDAP path to a zone uses the following format:

```
LDAP://[domain/]attr=name,[...],dc=domain_part,[...]
```

For example, if you use the default parent location for zones in the domain `arcade.com`, the LDAP path for the “default” zone is:

```
LDAP://cn=default,cn=zones,cn=Centrify,cn=program data,dc=arcade,dc=com
```

**Note:** The LDAP portion of the path is case sensitive. If you are unsure of the LDAP path for a zone, you can use the `adinfo` command on any computer in the zone to display the path.

## Exceptions

`GetZoneByPath` may throw one of the following exceptions:

- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.
- `ApplicationException` if the object cannot be located by the specified path.

## Example

The following code sample illustrates using this method in a script:

```
...  
string strUser = args[0];  
if (string.IsNullOrEmpty(strUser))  
{  
    Console.WriteLine("User DN cannot be empty.");  
}
```

• • • • •

```
        return;
    }
    // Obtain an active directory container object
    // Configure the test container
    DirectoryEntry objRootDSE = new DirectoryEntry("LDAP://rootDSE");
    DirectoryEntry objContainer = new DirectoryEntry("LDAP://" +
    strParent + "," + objRootDSE.Properties
    ["defaultNamingContext"].Value.ToString());
    string strContainerDN = objContainer.Properties
    ["DistinguishedName"].Value as string;
    // Create a CIMS object to interact with AD
    ICims cims = new Cims();
    // Note the lack of the cims.connect function.'
    // By default, this application will use the connection to the
    domain controller
    // and existing credentials from the computer already logged in.
    IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
    IHierarchicalZone;
    IUser objUser = cims.GetUserByPath(strUser);
    if (objUser == null)
    {
        Console.WriteLine("User " + strUser + " does not exist.");
        return;
    }
    ...
```

## IsForestConfigured

Indicates whether the Active Directory forest is configured with valid Centrify licenses.

### Syntax

```
bool IsForestConfigured()
```

### Return value

Returns true if the Active Directory forest is properly configured with at least one readable license, or false if no valid licenses are found.

### Exceptions

IsForestConfigured may throw one of the following exceptions:

- **COMException** if there is an LDAP error. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating



with Active Directory.

- `ApplicationException` if there is a global catalog server error. This exception may be thrown if the global catalog is not found or if LDAP errors occur during the discovery of the global catalog.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Check the Active Directory forest for licenses  
If not cims.IsForestConfigured then  
    wScript.Echo "Forest is not configured"  
End if  
...
```

## LoadLicenses

Returns all of the Centrify licenses installed on the connected domain.

### Syntax

```
ILicensesCollection LoadLicenses()
```

### Return value

The `Centrify.DirectControl.API.Licenses` object containing the collection of Centrify licenses installed on the connected domain.

### Discussion

This method returns the collection of all licenses in all of the license parent containers found in the forest and represented in the `LicensesCollection` object.

### Exceptions

`LoadLicenses` throws an `ApplicationException` if no license container is found or if any error occurs while accessing Active Directory.

## Example

The following code sample illustrates using this method in a script:

• • • • •

```
...  
'Get the collection of licenses  
If cims.IsForestConfigured = true then  
    set objLicense = LoadLicenses()  
end if  
...
```

## Password

Gets the logged-in user's password for connecting to the domain.

### Syntax

```
string Password {get;}
```

### Property value

The password used to connect to the domain.

### Example

The following code sample illustrates using this property in a script:

```
...  
'Connect to the domain controller Active Directory  
cims.Connect("paris.ajax.org","pierre","lesbleUs")  
'Display the password used to log on  
wScript.Echo "Current Password:" & cims.Password  
...
```

## Server

Gets the Active Directory domain controller name being used to establish the connection to the Active Directory domain.

### Syntax

```
string Server {get;}
```

### Property value

The domain controller name used to connect to the domain.

• • • • •

## Example

The following code sample illustrates using this property in a script:

```
...  
'Connect to the domain controller Active Directory  
cims.Connect("paris.ajax.org","pierre","lesbleus")  
'Display the domain controller name  
wScript.Echo "Connected to: " & cims.Server  
...
```

## UserName

Gets the Active Directory user name used to establish the connection to the Active Directory domain.

## Syntax

```
string UserName {get;}
```

## Property value

The Active Directory user name used to connect to the domain.

## Example

The following code sample illustrates using this property in a script:

```
...  
'Connect to the domain controller Active Directory  
cims.Connect("paris.ajax.org","pierre","lesbleus")  
'Display the user name  
wScript.Echo "Current User Credentials:" cims.UserName  
...
```

## Command

The Command class represents a command right.

## Syntax

```
public interface ICommand : IRight
```



## Methods

The `Command` class provides the following methods:

This method	Does this
<code>Commit</code>	Commits changes in the right to Active Directory. (Inherited from <code>Right</code> .)
<code>Delete</code>	Removes the right. (Inherited from <code>Right</code> .)

## Properties

The `Command` class provides the following properties:

This property	Does this
<code>AllowNestedExecution</code>	If <code>true</code> , allows the command to start another program or open a new shell.
<code>AuthenticationType</code>	Specifies the type of authentication required to run a command.
<code>CommandPattern</code>	Gets or sets the command string that is matched to identify the command.
<code>CommandPatternType</code>	Gets or sets the type of pattern used to match the command.
<code>Description</code>	Gets or sets the description of the right. (Inherited from <code>Right</code> .)
<code>DzdoRunASGroupList</code>	Gets or sets the list of groups allowed to run this command using <code>dzdo</code> .
<code>DzdoRunASUserList</code>	Gets or sets the list of users and groups allowed to run this command using <code>dzdo</code> .
<code>DzshRunASUser</code>	Gets or sets the user this command runs as when executed with <code>dzsh</code> .
<code>Guid</code>	Gets the GUID of the right.
<code>IsReadable</code>	Indicates whether the right is readable. (Inherited from <code>Right</code> .)
<code>IsResetVariables</code>	Resets or removes a default set of environment variables when running the command.
<code>IsWritable</code>	Indicates whether the right is writable. (Inherited from <code>Right</code> .)
<code>MatchPath</code>	Gets or sets the path for matching the specified command



This property	Does this
	name.
<b>Name</b>	Gets or sets the name of the right. (Inherited from <b>Right</b> .)
<b>PreserveGroupMembership</b>	Determines whether to retain the user's group membership while executing a command.
<b>UMask</b>	Gets or sets the <b>UMask</b> value to use for the command.
<b>VariablesToAdd</b>	Gets or sets the list of environment name-value pairs to add, such as <b>var1=a, var2=b, var3=c</b> .
<b>VariablesToKeepOrDelete</b>	Gets or sets the list of environment variables to keep or delete.
<b>weight</b>	Gets or sets the weight for this command.
<b>Zone</b>	Gets the zone to which this right belongs. (Inherited from <b>Right</b> .)

## Discussion

A command right controls who has permission to run a specific command in a zone.

## AllowNestedExecution

Determines whether the command is allowed to start another program or open a new shell.

### Syntax

```
bool AllowNestedExecution {get; set;}
```

### Property value

Set to true if the command is allowed to start another program or open a new shell. The default is true.

## AuthenticationType

Determines the type of authentication required to run a command.



## Syntax

```
AuthenticationType AuthenticationType {get; set;}
```

## Property value

The default value is to have no authentication required. If authentication is required, this property specifies the account used to authenticate before allowing use of the command right.

Possible values:

```
public enum AuthenticationType
{
    // No authentication required
    None = 0,
    // Authenticate using logged-on user password
    LoggedOnUserPassword,
    // Authenticate using target run-as user password
    RunasUserPassword
}
```

## CommandPattern

Gets or sets the command string that is matched to identify the command.

## Syntax

```
string CommandPattern {get; set;}
```

## Property value

The path to the command string.

## Discussion

Use the [CommandPatternType](#) property to get or set the type of command-pattern string matching to use.

## Exceptions

CommandPattern may throw the following exception:

- [ArgumentException](#) if the command pattern value is empty or null.



## Example

Glob expression: "rm ./\*"

Regular expression: "!finger sjohan | epage"

## CommandPatternType

Gets or sets the type of pattern used to match the command.

### Syntax

```
PatternType CommandPatternType {get; set;}
```

### Property value

The type of pattern-matching to use to identify the command.

Possible values:

```
public enum PatternType
{
    // Match using glob pattern
    Glob = 0,
    // Match using regular expression
    RegularExpression = 1
}
```

## DzdoRunAsGroupList

Gets or sets the list of groups allowed to run this command using dzdo.

### Syntax

```
string DzdoRunAsUserList {get; set}
```

### Property value

A comma-separated string of group names (for example, "group1, group2, group3"). If a value of "\*" is set, any group enabled for the zone can run the command.



## DzdoRunAsUserList

Gets or sets the list of users and groups allowed to run this command using dzdo.

### Syntax

```
string DzdoRunAsUserList {get; set}
```

### Property value

A comma-separated string of user and group names (for example, "user1,user2,group1"). If a value of "\*" is set, any user enabled for the zone can run the command.

### Discussion

If you don't specify a list in this property, by default only the root user can run the command.

## DzshRunAsUser

Gets or sets the user under which the command runs when executed using dzsh.

### Syntax

```
string DzshRunAsUser {get; set}
```

### Property value

The user name of a user authorized to run the sh command.

The default value is the current user.

## Guid

Gets the GUID of the command right.

### Syntax

```
Guid Guid {get;}
```



## Property value

The GUID of the command right.

## IsResetVariables

Determines whether to reset environment variables when running the command.

### Syntax

```
bool IsResetVariables {get; set;}
```

## Property value

Set true to reset environment values when the user runs the command.

## Discussion

The `dzdo.env_keep` configuration parameter in the `centrifdc.conf` file defines a set of environment variables to retain from the current user's environment when the command is run, regardless of whether the `IsResetVariables` property is `true` or `false`. When you set this property `true`, if you want to specify additional variables to retain from the user's environment, list the variables in the `VariablesToKeepOrDelete` property.

The `dzdo.env_delete` configuration parameter in the `centrifdc.conf` file defines a set of environment variables to delete from the current user's environment when the command is run, regardless of whether the `IsResetVariables` property is `true` or `false`. When you set this property `false`, if you want to specify additional environment variables to remove, list those variables in the `VariablesToKeepOrDelete` property.

## MatchPath

Gets or sets the match type for the path of the command.

### Syntax

```
string MatchPath {get; set;}
```



## Property value

The default value is "USERPATH".

Possible values to match:

- "USERPATH" Standard user path, starting with a forward slash (/).
- "SYSTEMPATH" Standard system path, starting with a forward slash (/).
- "SYSTEMSEARCHPATH" System search path, starting with a forward slash (/).
- A custom specific path, starting with a forward slash (/).
- All paths using a single asterisk (\*).

## PreserveGroupMembership

Determines whether to retain the user's group membership while executing the command.

### Syntax

```
bool PreserveGroupMembership {get; set;}
```

### Property value

Set true to preserve group membership.

The default is false.

## UMask

Determines the user file-creation mode mask (umask) value to use for the command.

### Syntax

```
string UMask {get; set;}
```

### Property value

The default Unix file permissions for new files created by the command, expressed as an octal number. For example, "764" or "077". The default

• • • • •

value is "077".

## Exceptions

UMask throws an `ArgumentException` if the specified permissions mask is not valid.

## VariablesToAdd

Gets or sets a comma-separated list of environment variable name-value pairs to add when the command is executed.

### Syntax

```
string VariablesToAdd {get; set;}
```

### Property value

A comma-separated string of name-value pairs (for example, "var1=a, var2=b, var3=c"). If a value of `null` or empty string is set, no name-value pairs are added. The default is `null`.

## Exceptions

`VariablesToAdd` throws an `ArgumentException` if `VariablesToAdd` contains an empty entry, the name, value, or name-value pair is invalid, or you listed duplicate variables.

## VariablesToKeepOrDelete

Get or set a comma-separated list of environment variables to keep or delete, depending on the value of the `IsResetVariables` property.

### Syntax

```
string VariablesToKeepOrDelete {get; set;}
```

### Property value

A comma-separated list of environment variables. The default is `null`.



## Discussion

This list is used by the `IsResetVariables` method to determine which variables should be kept or deleted when the command identified by this `Command` object is run.

## Exceptions

`VariablesToKeepOrDelete` throws an `ArgumentException` if the variable name is invalid or you specified duplicate variables.

## Weight

Gets or sets the weight of the command.

## Syntax

```
int weight {get; set;}
```

## Property value

The command priority. The default value is 0.

## Discussion

This number is when handling multiple matches for commands specified by wild cards. If commands specified by this command object match commands specified by another command object, the command object with the higher command priority prevails. The higher the value of the `weight` property, the higher the priority.

## Commands

The `Commands` class manages a collection of `Command` objects.

## Syntax

```
public interface ICommands
```



## Methods

The Commands class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Returns an enumeration of <code>Command</code> objects.

## GetEnumerator

Returns an enumeration of `Command` objects.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

The set of `Command` objects.

## Computer

The `Computer` class represents a Centrify-managed computer object.

### Syntax

```
public interface IComputer
```

## Methods

The `Computer` class provides the following methods:

This method	Does this
<code>Commit</code>	Commits changes to the computer object and saves them in Active Directory.
<code>Delete</code>	Removes the computer profile from Active Directory.
<code>GetDirectoryEntry</code>	Returns the directory entry for a computer object.
<code>Refresh</code>	Reloads the computer object data from the data in Active Directory.



## Properties

The Computer class provides the following properties:

This property	Does this
<b>AdsiInterface</b>	Gets the IADs interface for the computer object from Active Directory.
<b>ADSPath</b>	Gets the LDAP path to the computer object.
<b>AgentVersion</b>	Gets the version number of the Centrify agent as it is stored in Active Directory.
<b>CanonicalName</b>	Gets the canonical name of the computer object.
<b>IsOrphan</b>	Indicates whether the computer profile has a computer object associated with it.
<b>IsReadable</b>	Indicates whether the computer object is readable.
<b>IsWritable</b>	Indicates whether the computer object is writable.
<b>JBossEnabled</b>	Gets or sets the attribute that enables JBoss access for a computer account.
<b>Name</b>	Gets the computer name of the computer object.
<b>ProfileADSPath</b>	Gets the LDAP path to the computer profile associated with a computer object.
<b>SchemaVersion</b>	Gets the version of the Active Directory schema.
<b>TomcatEnabled</b>	Gets or sets the attribute that enables Tomcat access for a computer account.
<b>Version</b>	Gets the version number of the Active Directory schema.
<b>webLogicEnabled</b>	Gets or sets the attribute that enables WebLogic access for a computer account.
<b>webSphereEnabled</b>	Gets or sets the attribute that enables WebSphere access for a computer account.
<b>Zone</b>	Gets the zone associated with the <b>Computer</b> object.
<b>ZoneMode</b>	Gets the zone mode of the computer.

## Commit

Commits any changes or updates to the computer object and saves the changes to Active Directory.

## Syntax

```
void Commit()
```



## Discussion

When you use this method, it checks and validates the computer properties before saving the object in Active Directory. For example, before saving, the method validates that the computer name doesn't exceed the maximum length or contain invalid characters.

## Exceptions

`Commit` may throw one of the following exceptions:

- `ApplicationException` if the changes you are attempting to save contain one or more errors.
- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.
- `InvalidOperationException` if the computer profile cannot be found. This usually indicates that the computer is not in a zone.
- `UnauthorizedAccessException` if there are insufficient permissions to commit the Active Directory computer account object.

## Example

The following code sample illustrates using `Commit` for a computer object in a script:

```
...
set objZone = cims.GetZone("sierra.com/Performix/Zones/HongKong")
'Identify the computer account
set objComp = cims.GetComputer
("sierra.com/Performix/Computers/chu")
'Set a property and save the changes to the computer account
set objComp.WebSphereEnabled = true
objComp.Commit
...
```

## Delete

Removes the computer profile from Active Directory.

## Syntax

```
void Delete()
```



## Discussion

The computer profile is the service connection point associated with the computer object. This method does not remove the computer object itself from Active Directory.

## Exceptions

Delete may throw one of the following exceptions:

- `InvalidOperationException` if the computer profile cannot be found. This usually indicates that the computer is not in a zone.
- `UnauthorizedAccessException` if there are insufficient permissions to delete the Active Directory computer profile.

## Example

The following code sample illustrates using Delete for a computer object in a script:

```
...
set objZone = cims.GetZone("sierra.com/Performix/Zones/HongKong")
'Identify the computer account
set objComp = cims.GetComputerByPath("LDAP://CN=aix_fr03,
cn=Performix,CN=Computers,DC=sierra,DC=com")
'Delete the profile for the computer account
objComp.Delete
...
```

## GetDirectoryEntry

Returns the directory entry for the computer object.

### Syntax

```
DirectoryEntry GetDirectoryEntry ()
```

### Return value

The `DirectoryEntry` attribute of the computer object.

## Discussion

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-



based programs.

## Example

The following code sample illustrates using `GetDirectoryEntry` for a computer object in a script:

```
...
'Identify the computer account
IComputer computer = cims.GetComputerByPath("LDAP://CN=sage,
CN=Computers,DC=arcade,DC=com");
'Get the directory entry for the computer account
DirectoryEntry computerEntry = computer.GetDirectoryEntry();
'Rename the computer account
computerEntry.Rename("CN=sagebrush");
...
```

## Refresh

Reloads the Computer object data from the data in Active Directory.

## Syntax

```
void Refresh()
```

## Discussion

This method refreshes the computer properties in the cached object to ensure it is synchronized with the latest information in Active Directory.

## Exceptions

`Refresh` may throw one of the following exceptions:

- `InvalidOperationException` if the computer profile cannot be found. This usually indicates that the computer is not in a zone.
- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using `Refresh` for a computer object in a script:

• • • • •

```
...
set objZone = cims.GetZone("sierra.com/Performix/Zones/HongKong")
'Identify the computer account
Set objComp = cims.GetComputer
("sierra.com/Performix/Computers/chu")
'Set a property and save the changes to the computer account
Set objComp.WebSphereEnabled = true
objComp.Commit
'Refresh the computer object and display the result
objComp.Refresh
wScript.Echo objComp.WebSphereEnabled
...
```

## AdsInterface

Gets the IADs interface for the computer object from Active Directory.

### Syntax

```
IADs AdsInterface {get;}
```

### Property value

The IADs interface for the computer object.

### Example

The following code sample illustrates using `AdsInterface` for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
Set objComp = cims.GetComputerByPath("LDAP://CN=aixserver,
CN=computers,DC=sierra,DC=com")
'Display the LDAP path of the parent container
wScript.Echo objComp.AdsInterface.Parent
...
```

## ADsPath

Returns the LDAP path to the computer object.

### Syntax

```
string ADsPath {get;}
```

• • • • •

## Property value

The LDAP path to the computer object in the following format:

```
LDAP://CN=aixserver,CN=computers,DC=sierra,DC=com
```

Returns null if the computer profile is an orphan.

## Example

The following code sample illustrates using `ADsPath` for a computer object in a script:

```
...
set objZone = cims.GetZone("ajax.org/UNIX/Zones/")
'Identify the computer account
set objComp = cims.GetComputer("ajax.org/Computers/backup78")
wscript.Echo "LDAP path: " & objComp.ADsPath
...
```

## AgentVersion

Gets the version number of the Centrify agent as it is stored in Active Directory.

## Syntax

```
string AgentVersion {get;}
```

## Property value

The version number of the Centrify agent.

## Example

The following code sample illustrates using `AgentVersion` for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
set objComp = cims.GetComputerByPath("LDAP://CN=aixserver,
CN=computers,DC=sierra,DC=com")
wscript.Echo "Centrify Agent: " & objComp.AgentVersion
...
```

• • • • •

## CanonicalName

Gets the Active Directory canonical name of the computer object.

### Syntax

```
string CanonicalName {get;}
```

### Property value

The canonical name of the computer object.

### Example

The following code sample illustrates using `CanonicalName` for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
Set objComp = cims.GetComputerByPath("LDAP://CN=solaris10-dev,
CN=computers,DC=sierra,DC=com")
wscript.Echo "Canonical name: " & objComp.CanonicalName
...
```

## IsOrphan

Indicates whether the Centrify profile associated with a computer object is an orphan.

### Syntax

```
bool IsOrphan {get;}
```

### Property value

Returns `true` if the computer profile is an orphan, or `false` if the object is not an orphan.

### Discussion

A Centrify computer profile can become an orphan if the computer object it is associated with is deleted manually using Active Directory Users and



Computers or ADSI. Orphan data can consume disk space and reduce performance for directory services and should be removed.

## Example

The following code sample illustrates using `IsOrphan` for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Check for orphan profiles
for each computer in objZone.GetComputers
    If computer.IsOrphan then
        wScript.Echo computer.Name
    end if
next
...
```

## IsReadable

Indicates whether the data associated with the computer object is readable using the current permissions.

### Syntax

```
bool IsReadable {get;}
```

### Property value

Returns `true` if the computer object is readable, or `false` if the object is not readable.

### Discussion

This property returns a value of `true` if the user accessing the computer object in Active Directory has sufficient permissions to read its properties.

## Example

The following code sample illustrates using `IsReadable` for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
```

• • • • •

```
Set objComp = cims.GetComputer("ajax.org/Computers/backup78")
If not objComp.IsReadable then
    wScript.Echo "Computer account is not readable!"
end if
...
```

## IsWritable

Indicates whether the data associated with the computer object is writable by the current user.

### Syntax

```
bool IsWritable {get;}
```

### Property value

Returns `true` if the computer object is writable, or `false` if the object is not writable.

### Discussion

This property returns a value of `true` if the user accessing the computer object in Active Directory has sufficient permissions to change the computer object's properties.

### Example

The following code sample illustrates using `IsWritable` for a computer object in a script:

```
...
set objZone = cims.GetZone("ajax.org/UNIX/Zones/Pilot zone")
'Identify the computer account
Set objComp = cims.GetComputer("ajax.org/Computers/backup78")
If not objComp.IsWritable then
    wScript.Echo "You cannot save changes to this computer
account!"
end if
...
```

## JBossEnabled

Gets or sets the attribute that indicates whether the computer is enabled as a server for JBoss applications.

• • • • •

## Syntax

```
bool JBossEnabled {get; set;}
```

## Property value

Set to `true` if access to JBoss applications is enabled for the computer account, or `false` if access to JBoss applications is not enabled.

## Exceptions

`JBossEnabled` throws an `InvalidOperationException` if you try to set this property when the computer is not in a zone. For example, if you are using Centrify Express or have joined the domain using the `--workstation` option, you should not use this property.

## Example

The following code sample illustrates using `JBossEnabled` for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,CN=zones,
CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
Set objComp = cims.GetComputerByPath("LDAP://CN=aixserver,
CN=computers,DC=sierra,DC=com")
Set objComp.JBossEnabled = false
objComp.Commit
...
```

## Name

Gets the computer name of the computer object.

## Syntax

```
string Name {get;}
```

## Property value

The computer name of the computer object.

• • • • •

## Example

The following code sample illustrates using Name for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,CN=zones,
CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
Set objComp = cims.GetComputerByPath("LDAP://CN=magnolia,
CN=computers,DC=sierra,DC=com")
'Display the computer account name
wScript.Echo "Computer name: " & objComp.Name
...
```

## ProfileADsPath

Gets the LDAP path to a computer object's UNIX profile.

### Syntax

```
string ProfileADsPath {get;}
```

### Property value

The LDAP path for the computer profile associated with the computer object.

## Example

The following code sample illustrates using ProfileADsPath for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,CN=zones,
CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
Set objComp = cims.GetComputerByPath("LDAP://CN=magnolia,
CN=computers,DC=sierra,DC=com")
'Display the LDAP path to the computer's UNIX profile
wScript.Echo "LDAP path: " & objComp.ProfileADsPath
...
```

## SchemaVersion

Gets the version of the Active Directory data schema.

• • • • •

## Syntax

```
int SchemaVersion {get;}
```

## Property value

The version of the schema as an integer (int).

## Discussion

This property is used internally by the Centrify .NET module to identify the Active Directory schema being used.

**Note:** This property is designed for COM-based programs that support a 32-bit signed number. The property **Version** can be used in place of this property in .NET programs because .NET supports 64-bit signed numbers.

## Example

The following code sample illustrates using `SchemaVersion` for a computer object in a script:

```
...
set objZone = cims.GetZone("ajax.org/UNIX/Zones/Pilot zone
)
'Identify the computer account
Set objComp = cims.GetComputer("ajax.org/Computers/backup78")
wScript.Echo "Schema version: " & objComp.SchemaVersion
...
```

## TomcatEnabled

Determines whether the computer is enabled as a server for Tomcat applications.

## Syntax

```
bool TomcatEnabled {get; set;}
```

## Property value

Set to `true` if access to Tomcat applications is enabled for the computer account, or `false` if not.



## Exceptions

`TomcatEnabled` throws an `InvalidOperationException` if you try to set this property when the computer is not in a zone. For example, if you are using Centrify Express or have joined the domain using the `--workstation` option, you should not use this property.

## Example

The following code sample illustrates using `TomcatEnabled` for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
Set objComp = cims.GetComputerByPath("LDAP://CN=aixserver,
CN=computers,DC=sierra,DC=com")
Set objComp.TomcatEnabled = true
objComp.Commit
...
```

## Version

Gets the version number of the Active Directory data schema.

## Syntax

```
long version {get;}
```

## Property value

The version number of the Active Directory schema as a long integer value.

## Discussion

This property can be used only in .NET programs because .NET supports 64-bit signed numbers. This property cannot be used in COM-based programs. For COM-based programs, use the `SchemaVersion` property instead.

## Example

The following code sample illustrates using `version` for a computer object in a script:

```
...
//Create the top-level object
```

• • • • •

```
Cims cims = new Cims()
//Identify the computer account
IComputer computer = cims.GetComputer
("ajax.org/Computers/backup78")
Console.WriteLine "Schema version number: " + computer.Version
...
```

## WebLogicEnabled

Determines whether the computer is enabled as a server for WebLogic applications.

### Syntax

```
bool webLogicEnabled {get; set;}
```

### Property value

Set to `true` if access to WebLogic applications is enabled for the computer account or `false` if not.

### Exceptions

`webLogicEnabled` throws an `InvalidOperationException` if you try to set this property when the computer is not in a zone. For example, if you are using Centrify Express or have joined the domain using the `--workstation` option, you should not use this property.

### Example

The following code sample illustrates using `webLogicEnabled` for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
Set objComp = cims.GetComputerByPath("LDAP://CN=aixserver,
CN=computers,DC=sierra,DC=com")
Set objComp.WebLogicEnabled = true
objComp.Commit
...
```



## WebSphereEnabled

Determines whether the computer is enabled as a server for WebSphere applications.

### Syntax

```
bool webSphereEnabled {get; set;}
```

### Property value

Set to `true` if access to WebSphere applications is enabled for the computer account, or `false` if not.

### Exceptions

`webSphereEnabled` throws an `InvalidOperationException` if you try to set this property when the computer is not in a zone. For example, if you are using Centrify Express or have joined the domain using the `--workstation` option, you should not use this property.

### Example

The following code sample illustrates using `webSphereEnabled` for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,CN=zones,
CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
Set objComp = cims.GetComputerByPath("LDAP://CN=aixserver,
CN=computers,DC=sierra,DC=com")
Set objComp.WebSphereEnabled = false
objComp.Commit
...
```

## Zone

Gets or sets the zone object associated with the computer object.

### Syntax

```
IZone Zone {get; set;}
```



## Property value

The zone object for the zone of the computer account.

## Discussion

Each computer object can only be associated with one zone: the zone used to join the computer to its Active Directory domain. This property gets or sets the zone object for the zone to which the computer is currently joined.

## Exceptions

Zone may throw one of the following exceptions:

- `ApplicationException` if the zone you specify is null, an unsupported type, or already in use; or if you were trying to move the computer object to a new domain and the operation failed.
- `InvalidOperationException` if the computer is not zoned.

## Example

The following code sample illustrates using Zone for a computer object in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,CN=zones,
CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the computer account
set objComp = cims.GetComputerByPath("LDAP://CN=aixserver,
CN=computers,DC=sierra,DC=com")
'Display the zone name
wscript.Echo objComp.Zone.Name
...
```

## ZoneMode

Gets the zone mode of the computer; used internally by the .NET module.

## Syntax

```
ComputerZoneMode ZoneMode {get;}
```

## Property value

The zone mode of the computer.



Possible values:

```
public enum ComputerZoneMode
{
    // Unknown
    Unknown = 0,
    // The computer is joined to a zone
    Zoned = 1,
    // The computer is in workstation mode
    Workstation = 2,
    // The computer is in express mode
    Express = 4,
    // The computer is in null zone mode (no pam or nss)
    NullZone = 8,
};
```

## ComputerGroupUnixProfiles

Enumerates groups under a computer zone.

### Syntax

```
public interface IComputerGroupUnixProfiles : IGroupUnixProfiles
```

### Methods

The `ComputerGroupUnixProfiles` class provides the following methods:

This method	Does this
<code>Find</code>	Finds the group added to the computer.
<code>GetEnumerator</code>	Returns the enumeration of <code>GroupUnixProfile</code> objects. (Inherited from <code>GroupUnixProfiles</code> .)
<code>Refresh</code>	Reloads the cached <code>GroupUnixProfile</code> objects. (Inherited from <code>GroupUnixProfiles</code> .)

### Properties

The `ComputerGroupUnixProfiles` class provides the following properties:



This property	Does this
<code>Count</code>	Gets the number of <code>GroupUnixProfile</code> objects in this set. (Inherited from <code>GroupUnixProfiles</code> .)
<code>IsEmpty</code>	Determines whether the collection of UNIX group profiles is empty. (Inherited from <code>GroupUnixProfiles</code> .)

## Find

Finds the group added to the computer.

## Syntax

```
IHierarchicalGroup Find(IHierarchicalZoneComputer computer)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>computer</code>	The computer to search.

## Return value

The hierarchical group added to the computer, if any, or `null` if no group is found.

## ComputerRole

This class represents a computer role.

## Syntax

```
public interface IComputerRole
```

## Methods

The `ComputerRole` class provides the following methods:



This method	Does this
<code>AddAccessGroup</code>	Adds a user group to this computer role.
<code>AddRoleAssignment</code>	Adds an empty role assignment.
<code>AddUser</code>	Adds a user role assignment to this computer role.
<code>ClearCustomAttributes</code>	VBScript interface to clear the custom attributes for this class.
<code>Commit</code>	Saves changes.
<code>Delete</code>	Deletes this computer role.
<code>GetAccessGroup</code>	Gets a user group assigned to this computer role.
<code>GetAccessGroups</code>	Gets the user groups assigned to this computer role.
<code>GetCustomAttributeContainer</code>	Gets the directory entry for the parent container object for the custom attributes for this class.
<code>GetGroup</code>	Gets the AD computer group associated with this computer role.
<code>GetRoleAssignment</code>	Gets the role assignment for a specified role and user.
<code>GetRoleAssignmentById</code>	Gets the role assignment, given a GUID.
<code>GetRoleAssignments</code>	Returns all the user role assignments under this computer role.
<code>GetRoleAssignmentToAllADUsers</code>	Returns the role assignment given to all Active Directory users who have a specified role.
<code>GetRoleAssignmentToEveryone</code>	Returns the role assignment given to all users who have a specified role.
<code>GetUser</code>	Gets a user assigned to this computer role.
<code>GetUsers</code>	Gets the collection of users assigned to this computer role.
<code>ICustomAttributeContainer</code> <code>GetCustomAttributeContainer</code>	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class.
<code>SetCustomAttribute</code>	VBScript interface to set the custom attributes for this class.
<code>validate</code>	Validates the changes made to this computer role.

## Properties

The `ComputerRole` class provides the following properties:

This property	Does this
<code>CustomAttributes</code>	VBScript only: Gets or sets custom attributes for this computer role.
<code>Description</code>	Gets or sets the description of this computer role.



This property	Does this
<b>Group</b>	Gets or sets the AD computer group associated with this computer role.
<b>IsOrphan</b>	Indicates whether this computer role is an orphan.
<b>Name</b>	Gets or sets the name of this computer role.
<b>Zone</b>	Gets the zone of this computer role.

## Discussion

A computer role describes the intended use of a group of computers; for example, the set of computers dedicated as database servers. Each computer role has one associated Active Directory computer group, which identifies the computers that have that use. You can assign any number of users or user groups to a computer role, with each user or user group having the permissions necessary to perform a set of functions on computers in that computer role.

Note that, although there are conceptual similarities, a computer role is not a variety of access role. Whereas an access role is a set of permissions assigned to an Active Directory user or user group, a computer role defines the intended use of a group of computers. For example, the DBServer computer role might be associated with the Active Directory DatabaseServers computer group. Two user groups might be assigned to the DBServer computer role: DBUsers, which has the DatabaseUsers access role; and DBAdmins, which has the DatabaseAdmins role.

You can add custom attributes to role definitions and role assignments. For example, you might want to use a custom attribute to reference a ticket number associated with a specific type of access request, role definition, or temporary role assignment. Custom attributes are optional and you can use them to capture any kind of information that is meaningful to your organization.

You can add custom attributes when defining or modifying a role, defining or modifying a computer role, or when modifying role assignment properties.

The key point is that you can use the field for any type of information you might find useful. Customers most often want to reference a trouble/request ticket but the field can contain whatever you want.

## AddAccessGroup

Adds a user group to this computer role.



## Syntax

```
IAzRoleAssignment AddAccessGroup(DirectoryEntry group)
IAzRoleAssignment AddAccessGroup(SearchResult groupSr)
IAzRoleAssignment AddAccessGroup(string groupDn)
IAzRoleAssignment AddAccessGroup(IADsGroup groupIads)
```

## Parameters

Specify one of the following parameters when using this method.

Parameter	Description
group	The directory entry for the group you want to add.
groupSr	The directory entry for a group specified as a search result.
groupDn	The group specified as a distinguished name.
groupIads	The IADs interface to the group.

## Discussion

The `AddAccessGroup(DirectoryEntry group)` and `AddAccessGroup(SearchResult group)` methods are available only for .NET-based programs; call `AddRoleAssignment` for VBScript.

## Return value

The computer role assignment that includes the new group.

## Exceptions

`AddAccessGroup` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is null.
- `ApplicationException` if the parameter value is not a valid group or if it failed to create a role assignment because it cannot find the group.

## AddRoleAssignment

Adds an empty user role assignment to the computer role.

## Syntax

```
IRoleAssignment AddRoleAssignment()
```



## Return value

Returns an empty role assignment. This role assignment is not stored in Active Directory until you call the `RoleAssignment.Commit` method.

## Discussion

Any number of users can be assigned to a computer role and each of those users can have more than one role. Use this method to get an empty user role assignment for a computer role.

## AddUser

Adds a user to this computer role.

## Syntax

```
IAzRoleAssignment AddUser(DirectoryEntry user)
IAzRoleAssignment AddUser(SearchResult userSr)
IAzRoleAssignment AddUser(string userDn)
IAzRoleAssignment AddUser(IADsUser userIads)
```

## Parameters

Specify one of the following parameters when using this method.

Parameter	Description
<code>user</code>	The directory entry for the user you want to add.
<code>userSr</code>	The directory entry for a user specified as a search result.
<code>userDn</code>	The user specified as a distinguished name.
<code>userIads</code>	The IADs interface to the user.

## Return value

The computer role assignment that includes the new user.

## Discussion

The `AddUser(DirectoryEntry user)` and `AddAccessGroup(SearchResult user)` methods are available only for .NET-based programs. Call `AddRoleAssignment` for VBScript.



## Exceptions

AddUser may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `ApplicationException` if the parameter value is not a valid user or if it failed to create a role assignment because it cannot find the user.

## ClearCustomAttributes

Clears the custom attributes for this computer role.

### Syntax

```
void ClearCustomAttributes()
```

## Commit

Commits any changes or updates to a computer role and saves the changes to Active Directory.

### Syntax

```
void Commit()
```

## Discussion

This method does not validate changes. Call the `Validate` method before calling the `Commit` method.

## Exceptions

`Commit` throws an `ApplicationException` if it could not find the computer role, could not find authorization data for the role, or failed to commit the computer role due to a communication error.

## Delete

Marks the computer role for deletion from Active Directory.



## Syntax

```
void Delete()
```

## Exceptions

Delete throws an `ApplicationException` if it can't find the computer role, can't find authorization data for the zone, or failed to delete the role for another reason.

## GetAccessGroup

Gets a user group assigned to this computer role given a specific role.

## Syntax

```
IAzRoleAssignment GetAccessGroup(IRole role, DirectoryEntry group)
```

```
IAzRoleAssignment GetAccessGroup(IRole role, SearchResult groupSr)
```

```
IAzRoleAssignment GetAccessGroup(IRole role, string groupDn)
```

```
IAzRoleAssignment GetAccessGroup(IRole role, IADsGroup groupIAds)
```

## Parameters

Specify the following parameter when using this method:

Parameter	Description
role	The role of the group.

Specify one of the following parameters when using this method.

Parameter	Description
group	The directory entry for the group.
groupSr	The directory entry for a group specified as a search result.
groupDn	The group specified as a distinguished name.
groupIads	The IADs interface to the group.

## Return value

The computer role assignment that includes the specified group (`IAzRoleAssignment.TrusteeType==Group`).



## Discussion

Any number of user groups can be assigned to a computer role and each of those groups can have more than one role. Use this method to get the computer role assignment for a specific group and role.

The `GetAccessGroup(IRole role, DirectoryEntry group)` and `GetAccessGroup(IRole role, SearchResult groupSr)` methods are available only for .NET-based programs; call `AddRoleAssignment` for VBScript.

## Exceptions

`GetAccessGroup` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `ApplicationException` if the parameter value is not a valid group; or if it failed to get a role assignment because it cannot find the group.

## GetAccessGroups

Gets the user groups assigned to this computer role.

### Syntax

```
IRoleAssignments GetAccessGroups()
```

### Return value

The collection of computer role assignments. Enumerate this object to get all of the `IRoleAssignment` objects for this computer role that represent groups (`IRoleAssignment.TrusteeType==Group`).

## GetCustomAttributeContainer

Returns the directory entry for the parent container object for the custom attributes for this computer role.

### Syntax

```
ICustomAttributeContainer GetCustomAttributeContainer()
```



## Return value

The directory entry for the parent container object for the custom attributes for this computer role.

## Discussion

The `GetCustomAttributeContainer` method is available only for .NET-based programs.

## GetGroup

Returns the computer group associated with this computer role.

## Syntax

```
DirectoryEntry GetGroup()
```

## Return value

The directory entry for the computer group associated with the computer role.

## Discussion

The `GetGroup` method is available only for .NET-based programs; call `Group` for VBScript.

## GetRoleAssignment

Returns the user role assignment for a specified role and user.

## Syntax

```
IRoleAssignment GetRoleAssignment(IRole role, string dn)
```

## Parameters

Specify the following parameters when using this method.

Parameter	Description
<code>role</code>	The role for which you want a role assignment.
<code>dn</code>	The distinguished name of the user for which you want a role assignment.



## Return value

The role assignment for the specified role and user.

## Discussion

Any number of users can be assigned to a computer role and each of those users can have more than one role. Use this method to get the user role assignment for a specific user and role. To get the computer role assignment for a specific user, call the `GetUser` method.

## Exceptions

`GetRoleAssignment` throws an `ArgumentNullException` if one of the specified parameter values is `null`.

## GetRoleAssignmentById

Returns a role assignment, given a GUID.

## Syntax

```
IRoleAssignment GetRoleAssignmentById(Guid id)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>id</code>	The GUID of the role assignment.

## Return value

The role assignment that has the specified GUID.

## Discussion

This method returns a role assignment object given the GUID of the object.



## Exceptions

`GetRoleAssignmentById` throws an `ArgumentNullException` if the specified parameter value is `null`.

## GetRoleAssignments

Returns all the user role assignments under this computer role.

### Syntax

```
IRoleAssignments GetRoleAssignments()
```

### Return value

The collection of user role assignments for this computer role.

## GetRoleAssignmentToAllADUsers

Returns the role assignment given to all Active Directory users who have a specified role.

### Syntax

```
IRoleAssignment GetRoleAssignmentToAllADUsers(IRole role)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>role</code>	The user role for which you want the role assignment.

### Return value

The role assignment for the specified role.

## Exceptions

`GetRoleAssignmentToAllADUsers` throws an `ArgumentNullException` if the specified parameter value is `null`.



## GetRoleAssignmentToEveryone

Returns the role assignment given to all users who have a specified role.

### Syntax

```
IRoleAssignment GetRoleAssignmentToEveryone(IRole role)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
role	The user role for which you want the role assignment.

### Return value

The role assignment for the specified role.

### Discussion

This method returns the role assignment for everyone with the specified role, including local users and groups.

### Exceptions

`GetRoleAssignmentToEveryone` throws an `ArgumentNullException` if the specified parameter value is `null`.

## GetUser

Gets a user assigned to this computer role.

### Syntax

```
IAZRoleAssignment GetUser(IRole role, DirectoryEntry user)  
IAZRoleAssignment GetUser(IRole role, SearchResult usersr)  
IAZRoleAssignment GetUser(IRole role, string userDn)  
IAZRoleAssignment GetUser(IRole role, IADsUser userIads)
```



## Parameters

Specify the following parameter when using this method:

Parameter	Description
role	The role of the user.

Specify one of the following parameters when using this method.

Parameter	Description
user	The directory entry for the user you want to add.
userSr	The directory entry for a user specified as a search result.
userDn	The user specified as a distinguished name.
userIads	The IADs interface to the user.

## Return value

The computer role assignment for the specified user and role.

## Discussion

Any number of users can be assigned to a computer role and each of those users can have more than one role. Use this method to get the computer role assignment for a specific user and role. To get the user role assignment for a specific user, call the `GetRoleAssignment` method.

The `GetUser(IRole role, DirectoryEntry user)` and `GetUser(IRole role, SearchResult userSr)` methods are available only for .NET-based programs; call `User` for VBScript.

## Exceptions

`GetUser` may throw the following exceptions:

- `ApplicationException` if cannot find the computer role in the zone; if it cannot find the specified role; if it cannot find authorization information for the zone; or if it failed to get the role assignment for some other reason.
- `ArgumentNullException` if a specified parameter value is null.

## GetUsers

Gets the users assigned to this computer role.



## Syntax

```
IRoleAssignments GetUsers()
```

## Return value

The collection of computer roles. Enumerate this object to get all of the `IRoleAssignment` objects for this computer role that represent users (`IRoleAssignment.TrusteeType==User`).

## SetCustomAttribute

Sets the custom attribute for this computer role.

## Syntax

```
void SetCustomAttribute(string name, string value)
```

## Parameters

Specify the following parameters when using this method:

Parameter	Description
<code>name</code>	The name of the custom attribute.
<code>value</code>	The value of the custom attribute

## Return value

The collection of computer roles. Enumerate this object to get all of the `IRoleAssignment` objects for this computer role that represent users (`IRoleAssignment.TrusteeType==User`).

## Validate

Validates the data in the `ComputerRole` object before any changes are committed to Active Directory. The method validates the following:

- The computer role name is not empty.
- The computer role name does not duplicate an existing computer role name in the zone.



- The computer role exists in the zone.
- An Active Directory computer group has been specified for the computer role.
- The specified computer group exists.

If the `ComputerRole` object is marked for deletion, the method skips validation tests.

## Syntax

```
void validate()
```

## Exceptions

If the validation fails, `validate` may throw an `ApplicationException` with a message indicating which test failed.

## CustomAttributes

### Syntax

```
string CustomAttributes {get; set;}
```

### Property value

The custom attribute for this computer role.

## Description

Gets or sets the description of this computer role.

### Syntax

```
string Description {get; set;}
```

### Property value

A string describing the computer role.

• • • • •

## Group

### Syntax

```
string Group {get; set;}
```

### Property value

The Active Directory name of the computer group.

## IsOrphan

Indicates whether the computer role is an orphan.

### Syntax

```
bool IsOrphan {get;}
```

### Property value

Returns true if this computer role cannot link to its Active Directory group.

## Name

Gets or sets the name of the computer role.

### Syntax

```
string Name {get; set;}
```

### Property value

The name of the computer role.

## Zone

Gets the zone of the computer role.

### Syntax

```
IHierarchicalZone Zone {get;}
```



## Property value

The zone in which the computer role is defined.

# ComputerRoles

The ComputerRoles class manages a collection of computer roles.

## Syntax

```
public interface IComputerRoles
```

## Methods

The ComputerRoles class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Gets the enumerator you can use to enumerate all computer roles.

## Properties

The ComputerRoles class provides the following property:

This property	Does this
<code>IsEmpty</code>	Determines whether the collection is empty.

## GetEnumerator

Returns an enumeration of ComputerRole objects.

## Syntax

```
IEnumerator GetEnumerator()
```

## Return value

Returns an enumerator you can use to list all the ComputerRole objects.



## IsEmpty

Indicates whether the collection of computer roles is empty.

### Syntax

```
bool IsEmpty {get;}
```

### Property value

Returns true if there are no `ComputerRole` objects in the `ComputerRoles` object.

## Computers

The `Computers` class manages a collection of `Computer` objects.

### Syntax

```
public interface IComputers
```

### Methods

The `Computers` class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Gets an enumerator you can use to enumerate all computer objects.

### Properties

The `ComputerRoles` class provides the following property:

This property	Does this
<code>IsEmpty</code>	Determines whether the collection is empty.

• • • • •

## GetEnumerator

Returns an enumeration of Computer objects.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

Returns an enumerator you can use to list all the Computer objects.

## IsEmpty

Determines whether the collection of computer objects is empty.

### Syntax

```
bool IsEmpty {get;}
```

### Property value

Returns true if there are no Computer objects in the Computers object.

## ComputerUserUnixProfiles

The ComputerUserUnixProfiles class manages a collection of UserUnixProfile objects that represent computer users.

### Syntax

```
public interface IComputerUserUnixProfiles : IUserUnixProfiles
```

### Methods

The ComputerUserUnixProfiles class provides the following methods:



This method	Does this
<code>Find</code>	Finds the user added to the specified computer.
<code>GetEnumerator</code>	Returns the enumeration of user profiles. (Inherited from <code>UserUnixProfiles</code> .)
<code>Refresh</code>	Reloads the cached user UNIX profiles. (Inherited from <code>UserUnixProfiles</code> .)

## Properties

The `ComputerUserUnixProfiles` class provides the following properties:

This property	Does this
<code>Count</code>	Gets the number of <code>UserUnixProfile</code> objects in this collection. (Inherited from <code>UserUnixProfiles</code> .)
<code>IsEmpty</code>	Determines whether the collection is empty. (Inherited from <code>UserUnixProfiles</code> .)

## Find

Finds the UNIX user profile of the user of a specified computer.

## Syntax

```
IHierarchicalUser Find(IHierarchicalZoneComputer computer)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>computer</code>	The computer to search.

## Return value

The UNIX user profile of the user of the specified computer; `null` if none exists. If there is more than one user, the first UNIX profile found is returned.



## CustomAttributesContainer

The `CustomAttributesContainer` class contains a collection of custom attributes. Available for .NET only.

### Methods

The `CustomAttributesContainer` class provides the following methods:

This method	Does this
<code>ClearCustomAttributes</code>	Clears the custom attributes.
<code>ICustomAttributes</code> <code>GetCustomAttributes</code>	Interface to return the custom attributes.
<code>SetCustomAttribute</code>	Interface to set the custom attributes for this class.
<code>ValidateCustomAttributes</code>	Validates the custom attributes.

### ICustomAttributes GetCustomAttributes

Gets the `CustomAttributes`.

### Syntax

```
ICustomAttributes GetCustomAttributes(IHierarchicalZoneComputer  
computer)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>computer</code>	The computer to search.

### Return value

The custom attributes of the specified computer; `null` if none exists.



## ValidateCustomAttributes

Validates the CustomAttributes.

### Syntax

```
validateCustomAttributes(IHierarchicalZoneComputer computer)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
computer	The computer to search.

### Return value

A boolean value; true if the custom attributes are valid. Otherwise, false.

## CustomAttributes

The CustomAttributes class contains a set of custom attributes. Available for .NET only.

### Methods

The CustomAttributeContainer class provides the following methods:

This method	Does this
IEnumerator GetEnumerator	Interface to enumerate the custom attributes.

### GetEnumerator

Returns an enumeration of CustomAttributes objects.

### Syntax

```
IEnumerator GetEnumerator()
```



## Return value

The set of `CustomAttributes` objects.

## CustomAttributes

The `CustomAttributes` class contains a custom attribute. Available for .NET only.

## Properties

The `CustomAttributes` class provides the following properties:

This property	Does this
<code>Name</code>	The name of the custom attribute, specified as a string.
<code>Value</code>	The value of the custom attribute, specified as a string.

## Group

Centrify uses existing Active Directory groups to manage the members of UNIX groups.

## Syntax

```
public interface IGroup
```

## Discussion

The `Group` class provides access to methods and properties that enable UNIX group profiles to be linked to Active Directory groups and that you can use to manage UNIX profiles associated with Active Directory groups. The additional UNIX-specific attributes that make up the UNIX profile for a group are stored and managed within the `GroupUnixProfile` object.



## Methods

The Group class provides the following methods:

This method	Does this
<code>AddUnixProfile</code>	Adds a new UNIX group profile to a zone.
<code>Commit</code>	Validates and saves changes to the <b>Group</b> object in Active Directory.
<code>CommitWithoutCheck</code>	Saves changes to the <b>Group</b> object in Active Directory without performing any validation.
<code>GetDirectoryEntry</code>	Returns the directory entry for an Active Directory group object from Active Directory.
<code>GetRoleAssignmentsFromDomain</code>	Returns the collection of all role assignments for a group in a specified domain.
<code>GetRoleAssignmentsFromForest</code>	Returns the collection of all role assignments for a group in a specified forest.
<code>Refresh</code>	Reloads the <b>Group</b> object data from the data in Active Directory.

## Properties

The Group class provides the following properties:

This property	Does this
<code>AdsiInterface</code>	Gets the IADs interface for an Active Directory group.
<code>ADSPATH</code>	Gets the LDAP path for an Active Directory group.
<code>ID</code>	Gets the unique identifier for an Active Directory group.
<code>UnixProfiles</code>	Gets the <b>GroupUnixProfiles</b> object associated with an Active Directory group.

## AddUnixProfile

Adds a new UNIX group profile to a zone.

### Syntax

```
IGroupUnixProfile AddUnixProfile(IZone zone, int gid, string name)
```

```
IGroupUnixProfile AddUnixProfile(IZone zone, long gid, string name)
```



## Parameters

Specify the following parameters when using this method.

Parameter	Description
zone	The individual zone to which you are adding a new UNIX group profile.
gid	The GID of the new UNIX group profile.
name	The name of the new UNIX group profile.

## Return value

The UNIX group object created.

## Discussion

The UNIX group profile includes the group name and the numeric group identifier (GID).

**Note:** There are two versions of this method: one designed for COM-based programs that supports a 32-bit signed number for the `gid` argument and one designed for .NET-based programs that allows a 64-bit signed number for the `gid` argument.

## Exceptions

`AddUnixProfile` may throw one of the following exceptions:

- `ArgumentNullException` if the `zone` parameter value is `null`.
- `NotSupportedException` if the specified zone has an unrecognized schema.

## Example

The following code sample illustrates using `AddUnixProfile` in a script:

```
...
if (objGroup.UnixProfiles.Find(objZone) == null)
{
    long next_gid = 10000; // use 10000 as default gid
    // Get the next available GID for this zone
    if (objZone.NextAvailableGID >= 0)
    {
        next_gid = objZone.NextAvailableGID;
    }
    // Add this zone to the group
    objGroupUnixProfile = objGroup.AddUnixProfile(objZone, next_gid,
    strUnixGroup);
}
```

• • • • •

```
// Save  
objGroupUnixProfile.Commit();  
...
```

## Commit

Commits any changes or updates to the group object and saves the changes to Active Directory.

## Syntax

```
void Commit()
```

## Discussion

When you use this method, it checks and validates the data before saving it in Active Directory. Before saving, the method validates the following:

- The group name is a valid string that contains only letters (upper- or lowercase), numerals 0 through 9, and the hyphen (-) and underscore (\_) characters.
- The GID value is a positive integer. Negative numbers are not allowed.
- The group name does not duplicate an existing group name.

## Exceptions

`Commit` may throw one of the following exceptions:

- `ApplicationException` if any field in the UNIX group profile is invalid.
- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.
- `UnauthorizedAccessException` if you have insufficient permissions to commit the group object to Active Directory.

## Example

The following code sample illustrates using `Commit` in a script:

```
...  
if (objGroup.UnixProfiles.Find(objZone) == null)  
{  
    Console.WriteLine( strGroup + " was not a member of " +
```

• • • • •

```
strZone);  
    return;  
}  
else  
    // Remove group  
    objGroup.RemoveGroupUnixProfile(objZone);  
    objGroup.Commit();  
}  
...
```

## CommitWithoutCheck

Commits any changes or updates to the Group object and saves the changes to Active Directory without validating any of the data fields.

### Syntax

```
void CommitWithoutCheck()
```

### Discussion

Because this method does not perform any validation checking, it commits changes faster than the `Group.Commit` method.

### Exceptions

`CommitWithoutCheck` may throw one of the following exceptions:

- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.
- `UnauthorizedAccessException` if you have insufficient permissions to commit the group object to Active Directory.

### Example

The following code sample illustrates using `CommitWithoutCheck` in a script:

```
...  
'Identify the zone you want to work with  
set zone = cims.GetZone("ajax.org/UNIX/Zones/eur007")  
'Identify the Active Directory group  
set group = cims.GetGroupByPath("LDAP://CN=Subcontractors,  
CN=EuropeanDiv,DC=ajax,DC=org")  
'Set the UNIX profile associated with the group  
group.SetGroupUnixProfile(zone, 8234, "subs")
```

• • • • •

```
'Update Active Directory without validation  
group.CommitWithoutCheck  
...
```

## GetDirectoryEntry

Returns a `DirectoryEntry` object for the Active Directory group account from Active Directory.

### Syntax

```
DirectoryEntry GetDirectoryEntry()
```

### Return value

The directory entry for the UNIX group profile associated with the Active Directory group.

### Discussion

The `DirectoryEntry` object represents the service connection point associated with the group in the zone.

**Note:** This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

### Exceptions

`GetDirectoryEntry` throws an `ApplicationException` if the directory object cannot be retrieved—for example, if it has not been committed.

### Example

The following code sample illustrates using `GetDirectoryEntry` in a script:

```
...  
//Identify the group you want to work with  
IGroup group = cims.GetGroup("LDAP://CN=oracle1, CN=Users,  
DC=ajax, DC=org");  
//Get the directory entry  
DirectoryEntry groupEntry = group.GetDirectoryEntry();  
//Rename the group  
groupEntry.Rename("CN=oracle_dbas");  
...
```

• • • • •

## GetRoleAssignmentsFromDomain

Returns the collection of all role assignments explicitly assigned to a specified group—regardless of whether the role assignment is in a zone, computer-specific (computer override) zone, or computer role—within a specified domain.

### Syntax

```
IRoleAssignments GetRoleAssignmentsFromDomain(string domain)
```

### Parameters

Specify the following parameter when using this method:

Parameter	Description
domain	The domain to search for the group's role assignments.

### Return value

A collection of role assignment objects representing all of the role assignments explicitly assigned to this group in the specified domain or in the currently joined domain.

### Discussion

This method only returns role assignments explicitly assigned to the group. The method does not expand the group membership or return role assignments for groups nested under the specified group.

The method will look for stored credentials to access the specified domain. If there are no stored credentials, it uses the default credentials for the current user.

If you don't specify a domain by passing an empty string ("") to the method, the method returns role assignments from the currently joined domain.

### Example

The following code sample illustrates using `GetRoleAssignmentsFromDomain` in a script:

```
...  
# New Cims object  
$cims = New-Object ("Centrify.DirectControl.API.Cims");  
# Get IGroup object  
$objGroupDn = "CN=group1,CN=Users,DC=domain,DC=com";
```



```
$objGroup = $cims.GetGroup($objGroupDn);  
# Get role assignments from domain  
$objGroup.GetRoleAssignmentsFromDomain("domain.com")  
...
```

## GetRoleAssignmentsFromForest

Returns the collection of all role assignments explicitly assigned to a specified group—regardless of whether the role assignment is in a zone, computer-specific (computer override) zone, or computer role—within a specified forest.

### Syntax

```
IRoleAssignments GetRoleAssignmentsFromForest(string forest)
```

### Parameters

Specify the following parameter when using this method:

Parameter	Description
forest	The forest to search for the group's role assignments.

### Return value

A collection of role assignments objects representing all of the role assignments explicitly assigned to this group in the specified forest or in the currently joined forest.

### Discussion

This method only returns role assignments explicitly assigned to the group. The method does not expand the group membership or return role assignments for groups nested under the specified group.

The method will look for stored credentials to access the specified forest. If there are no stored credentials, it uses the default credentials for the current user.

If you don't specify a forest by passing an empty string ("" ) to the method, the method returns role assignments from the currently joined forest.



## Example

The following code sample illustrates using `GetRoleAssignmentsFromForest` in a script:

```
...  
# New Cims object  
$cims = New-Object ("Centrify.DirectControl.API.Cims");  
# Get IGroup object  
$objGroupDn = "CN=group1,CN=Users,DC=domain,DC=com";  
$objGroup = $cims.GetGroup($objGroupDn);  
# Get role assignments from forest  
$objGroup.GetRoleAssignmentsFromForest("forest.com")  
...
```

## Refresh

Reloads the group object data from the data in Active Directory.

## Syntax

```
void Refresh()
```

## Discussion

This method refreshes the group information in the cached object to ensure it is synchronized with the latest information in Active Directory.

## Example

The following code sample illustrates using `Refresh` in a script:

```
...  
'Identify the zone you want to work with  
set zone = cims.GetZone("ajax.org/UNIX/Zones/eur007")  
'Identify the Active Directory group  
set group = cims.GetGroupByPath("LDAP://CN=Subcontractors,  
CN=EuropeanDiv,DC=ajax,DC=org")  
'Modify the UNIX profile associated with the group  
group.SetGroupUnixProfile(zone, 8234, "subcon07")  
group.Commit  
'Reload the group object from Active Directory  
group.Refresh  
wScript.Echo "Group Unix Profile Name: " & group.Name  
...
```

• • • • •

## AdsInterface

Gets the IADsGroup interface for the group object from Active Directory.

### Syntax

```
IADsGroup AdsInterface {get;}
```

### Property value

The IADsGroup interface for the group object.

### Example

The following code sample illustrates using AdsInterface in a script:

```
...
'Identify the zone you want to work with
set objZone = cims.GetZoneByPath("LDAP://cn=eur007,cn=Zones,
cn=UNIX,dc=ajax,dc=org")
'Identify the Active Directory group
set objGroup = cims.GetGroupByPath
("LDAP://cn=Subcontractors,cn=EuropeanDiv,dc=ajax,dc=org")
'Get ADSI interface associated with the group
Set objAdsi = objGroup.AdsInterface
...
```

## ADsPath

Gets the LDAP path for the specified Active Directory group.

### Syntax

```
string ADsPath {get;}
```

### Property value

The LDAP path for the Active Directory group object.

### Example

The following code sample illustrates using ADsPath for a group in a script:

```
...
'Identify the zone you want to work with
set objZone = cims.GetZone("sierra.com/program
data/centrify/zones/market research")
```

• • • • •

```
'Identify the Active Directory group
Set objGroup = cims.GetGroup("sierra.com/Groups/Managers")
'Display the LDAP for the specified group
wScript.Echo "LDAP path: " & objGroup.ADsPath
...
```

## ID

Gets the unique identifier for the specified Active Directory group.

### Syntax

```
string ID {get;}
```

### Property value

The GUID for the specified Active Directory group.

### Example

The following code sample illustrates using ID for a group in a script:

```
...
'Identify the zone you want to work with
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the Active Directory group
Set objGroup = cims.GetGroupByPath("LDAP://CN=managers,
CN=Groups,DC=sierra,DC=com")
'Display the ID for the specified group
wScript.Echo "Unique ID: " & objGroup.ID
...
```

## UnixProfiles

Gets the GroupUnixProfiles object associated with a specified Active Directory group.

### Syntax

```
IGroupUnixProfiles UnixProfiles {get;}
```

• • • • •

## Property value

The `GroupUnixProfiles` object associated with the specified Active Directory group.

## Discussion

The `GroupUnixProfiles` object contains information about the collection of group profiles associated with the Active Directory group in different zones.

## Example

The following code sample illustrates using `UnixProfiles` in a script:

```
...
//Create a CIMS object to interact with AD
ICims cims = new Cims();
//Note the lack of the cims.connect function.
//By default, this application will use the connection to the
domain controller
//and existing credentials from the computer already logged in.
//Get the group object
IGroup objGroup = cims.GetGroupByPath(strGroup);
//Get the zone object
IZone objZone = cims.GetZoneByPath("cn=" + strZone + "," +
strContainerDN);
// Determine if the specified group is already a member of the
zone.
// This method will either return a blank objGroupUnixProfile
// or one containing data
if (objGroup.UnixProfiles.Find(objZone) == null)
{
    Console.WriteLine( strGroup + " was not a member of " +
strZone);
    return;
}
else
{
    // Remove group
    objGroup.RemoveGroupUnixProfile(objZone);
    objGroup.Commit();
}
...
```

## GroupUnixProfile

The `GroupUnixProfile` class manages the UNIX group profile information of an Active Directory group or a local group in a given zone.



## Syntax

```
public interface IGroupUnixProfile
```

## Discussion

An Active Directory or local group's zone-specific UNIX profile includes the numeric GID value and profile name directory.

## Methods

The `GroupUnixProfile` class provides the following methods:

This method	Does this
<code>Commit</code>	Commits changes to the <code>GroupUnixProfile</code> object to Active Directory.
<code>Delete</code>	Marks the UNIX group profile object for deletion from Active Directory.
<code>GetDirectoryEntry</code>	Returns the <code>DirectoryEntry</code> for a UNIX group profile from Active Directory.
<code>Refresh</code>	Reloads the <code>GroupUnixProfile</code> object data from the data in Active Directory.
<code>Validate</code>	Validates data in the <code>GroupUnixProfile</code> object before the changes are committed to Active Directory.

## Properties

The `GroupUnixProfile` class provides the following properties:

This property	Does this
<code>ADSPATH</code>	Gets the LDAP path to the UNIX group profile.
<code>Cims</code>	Gets the Cims data for the group profile.
<code>Group</code>	Gets the Active Directory group to which the <code>GroupUnixProfile</code> object belongs (Active Directory groups only).
<code>GroupID</code>	Gets or sets the numeric group identifier (GID) for the group profile.
<code>ID</code>	Gets the unique identifier for the UNIX group profile.
<code>IsForeign</code>	Indicates whether the UNIX profile for a group is in a different forest than its corresponding Active Directory group (Active



This property	Does this
	Directory groups only).
<b>IsMembershipRequired</b>	Determines whether an Active Directory group is a required group (Active Directory groups only).
<b>IsOrphan</b>	Indicates whether this UNIX group profile is an orphan (Active Directory groups only).
<b>IsReadable</b>	Indicates whether the Active Directory object is readable.
<b>IsSFU</b>	Indicates whether this UNIX group is an SFU zone profile (Active Directory groups only).
<b>IsWritable</b>	Indicates whether the Active Directory object is writable.
<b>Members</b>	Gets or sets the local group members of the UNIX group profile (local groups only).
<b>Name</b>	Gets or sets the group name of the UNIX group profile.
<b>ProfileState</b>	Gets or sets the profile state of the local group profile (local groups only).
<b>Type</b>	Gets the type of the UNIX group profile.
<b>UnixEnabled</b>	Determines whether the UNIX information is enabled.
<b>Zone</b>	Gets the zone object for the current <code>GroupUnixProfile</code> object.

## Commit

Commits any changes or updates to the `GroupUnixProfile` object and saves the changes to Active Directory.

## Syntax

```
void Commit()
```

## Discussion

This method commits to Active Directory any new or changed values in the group UNIX profile. If an object is marked for deletion, calling this method completes the operation and deletes the object from Active Directory. The method also increments the next available group identifier (GID) by one, if applicable and permitted. The method does not validate the data before saving it in Active Directory.

## Exceptions

`Commit` may throw one of the following exceptions:



- `ApplicationException` if it failed to get the directory entry of the group profile.
- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.
- `ObjectAlreadyExistsException` if the method receives a `COMException` with an LDAP object already exists error code.

## Example

The following code sample illustrates using `Commit` in a script:

```
...  
'Identify the zone you want to work with  
set objZone = cims.GetZoneByPath("LDAP://cn=onsite,cn=Zones,  
cn=UNIX,dc=arcade,dc=com")  
'Identify the Active Directory group  
Set objGroup = cims.GetGroupByPath("CN=escalation,CN=support,  
DC=arcade,DC=com")  
'Remove the membership requirement for the group  
Set objGroup.IsMembershipRequired = false  
'Save the changes in Active Directory  
objGroup.commit  
...
```

## Delete

Marks the UNIX group profile object for deletion from Active Directory.

## Syntax

```
void Delete()
```

## Discussion

This method does not delete the group profile—it only marks it for deletion. After you mark an object for deletion, you must call the `Commit` method to complete the operation.

## Example

The following code sample illustrates using `Delete` in a script:

```
...  
'Identify the zone you want to work with
```

• • • • •

```
set objZone = cims.GetZoneByPath("LDAP://cn=eur007,cn=Zones,
cn=UNIX,dc=ajax,dc=org")
'Get the UNIX group profile you want to delete
set objProfile = objZone.GetGroupUnixProfileByGid("905")
objProfile.Delete
...
```

## GetDirectoryEntry

Returns an instance of the directory entry for the group's UNIX profile from Active Directory.

### Syntax

```
DirectoryEntry GetDirectoryEntry ()
```

### Return value

The `DirectoryEntry` object for the UNIX group profile associated with the Active Directory group.

### Discussion

The `DirectoryEntry` object represents the service connection point associated with the group in the zone.

**Note:** This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

### Example

The following code sample illustrates using `GetDirectoryEntry` in a script:

```
...
//Identify the zone you want to work with
IZone zone = cims.GetZone("ajax.org/UNIX/Zones/NW_Support")
//Display the access control list
foreach (IGroupUnixProfile gpProfile in
zone.GetGroupUnixProfiles())
{
    //Get the directory entry
    DirectoryEntry scp = gpProfile.GetDirectoryEntry();
    Console.WriteLine
(scp.ObjectSecurity.GetSecurityDescriptorSddlForm
(AccessControlSections.Access));
}
...
```

• • • • •

## Refresh

Reloads the GroupUnixProfile object data from the data in Active Directory.

### Syntax

```
void Refresh()
```

### Discussion

This method refreshes the group profile information in the cached object to ensure it is synchronized with the latest information in Active Directory.

### Example

The following code sample illustrates using Refresh in a script:

```
...  
'Identify the zone you want to work with  
set objZone = cims.GetZoneByPath("LDAP://cn=eur007,cn=Zones,  
cn=UNIX,dc=ajax,dc=org")  
'Get the UNIX group profile you want to work with  
set objProfile = objZone.GetGroupUnixProfileByGid("905")  
'Reload the group profile object from Active Directory  
objProfile.Refresh  
...
```

## Validate

Validates the data in the GroupUnixProfile object before any changes are committed to Active Directory.

### Syntax

```
void validate()
```

### Discussion

The method validates the following:

- The group name is a valid string that can contain only letters (upper- or lowercase), numerals 0 through 9, and the hyphen (-) and underscore (\_) characters.
- The GID value is a positive integer. Negative numbers are not allowed.



- The group profile does not duplicate an existing group identifier (GID) or group name.

If the `GroupUnixProfile` object is marked for deletion, the method skips validation tests.

## Exceptions

`validate` throws an `ApplicationException` if any field in the UNIX group profile is invalid.

## Example

The following code sample illustrates using `validate` in a script:

```
...
'Identify the zone you want to work with
set objZone = cims.GetZoneByPath("LDAP://cn=eur007,cn=Zones,
cn=UNIX,dc=ajax,dc=org")
'Get the UNIX group profile you want to work with
set objProfile = objZone.GetGroupUnixProfileByGid("905")
'Validate the UNIX profile associated with the group
objProfile.Validate
...
```

## ADsPath

Gets the LDAP path to the UNIX group profile object.

## Syntax

```
string ADsPath {get;}
```

## Property value

The LDAP path to the UNIX group profile.

## Example

The following code sample illustrates using `ADsPath` in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrifly,CN=program data,DC=sierra,DC=com")
'Identify the Active Directory group
set objGroup = cims.GetGroupByPath("LDAP://CN=managers,CN=groups,
DC=sierra,DC=com")
'Get the UNIX profile for the group in the zone
```

• • • • •

```
set objGroupUnixProfile = objGroupUnixProfiles.Find(objZone)
'Display the LDAP path for this group profile
wScript.Echo "LDAP Path: " & objGroupUnixProfile.ADsPath
...
```

## Cims

Gets the Cims object for the group profile.

### Syntax

```
Cims Cims {get;}
```

### Property value

The Cims object for the group profile.

### Discussion

This property serves as a shortcut for retrieving data.

### Example

The following code sample illustrates using Cims in a script:

```
...
function doThings(gp2)
    set objZone2 = gp2.Cims.GetZone("ajax.org/Zones/test")
    gp2.Group.AddUnixProfile
objZone2,objProfile.Gid,objProfile.Name
end function
set cims = CreateObject("Centrify.DirectControl.Cims3")
set objZone = cims.GetZone("ajax.org/Zones/default")
for each gp2 in objZone.GetGroupUnixProfiles
    doThings gp2
next
...
```

## Group

Gets the Active Directory group object associated with the specified GroupUnixProfile object.

### Syntax

```
IGroup Group {get;}
```

• • • • •

## Property value

The Active Directory group object associated with the UNIX group profile.

## Example

The following code sample illustrates using `Group` in a script:

```
...
'Identify the zone you want to work with
set objZone = cims.GetZone("ajax.org/UNIX/Zones/eur007")
'Get the UNIX group profile you want to work with
set objProfile = objZone.GetGroupUnixProfileByGid("905")
'Display the LDAP path to the profile's Active Directory group
wScript.Echo "LDAP path: " & objProfile.Group.ADsPath
...
```

## GroupID

Gets the numeric group identifier (GID) for the group profile or sets a new GID for the specified Active Directory group in the specified zone.

## Syntax

```
long GroupID {get; set;}
```

## Property value

The numeric value of the UNIX GID for the UNIX profile in the zone.

## Discussion

This property supports a 64-bit signed number for .NET modules.

## Exceptions

`GroupID` throws and `InvalidOperationException` if the GID is null (that is, there is only a partial profile).

## ID

Gets the unique identifier for the UNIX group profile from Active Directory.

• • • • •

## Syntax

```
string ID {get;}
```

## Property value

The unique identifier for this UNIX group profile.

## Example

The following code sample illustrates using ID in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,CN=zones,
CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the Active Directory group
Set objGroup = cims.GetGroupByPath("LDAP://CN=managers,CN=groups,
DC=sierra,DC=com")
'Get the UNIX profile for the group in the zone
set objGroupUnixProfile = objGroupUnixProfiles.Find(objZone)
'Display the unique ID for this group
wscript.Echo "Unique ID: " & objGroupUnixProfile.ID
...
```

## IsForeign

Indicates whether the corresponding Active Directory group for a UNIX profile is in a different Active Directory forest than the forest associated with the group profile in the zone.

## Syntax

```
bool IsForeign {get;}
```

## Property value

Returns true if the UNIX profile is associated with an Active Directory group in a different forest.

## Discussion

If the Active Directory group is in a different forest than the one associated with a top-level Centrify data object (Cims object), the property returns true.

**Note:** This property is always false for newly-created groups before the group object is committed to Active Directory.



## Example

The following code sample illustrates using `IsForeign` in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Check the forest for groups in the zone  
For each profile in objZone.GetGroupUnixProfiles  
if profile.IsForeign then  
    wScript.Echo profile.Name  
end if  
next  
...
```

## IsMembershipRequired

Determines whether the Active Directory group is a required group for its members.

### Syntax

```
bool IsMembershipRequired {get; set;}
```

### Property value

Returns `true` if the UNIX profile associated with an Active Directory group is marked as a required group.

### Discussion

If this property is `true`, users cannot use the `adsetgroups` command to remove the group from the currently active set of groups. If this property is `false`, users who are members of the group can add or remove the group from their list of active groups at any time.

For more information about making a group required, see the *Administrator's Guide for Linux and UNIX*.

### Exceptions

`IsMembershipRequired` throws an `InvalidOperationException` if there is only a partial profile.

• • • • •

## Example

The following code sample illustrates using `IsMembershipRequired` in a script:

```
...
Set objZone = cims.GetZoneByPath("LDAP://CN=research,CN=zones,
CN=centrify,CN=program data,DC=sierra,DC=com")
'Get the UNIX group profile you want to work with
set objProfile = objZone.GetGroupUnixProfileByGid("905")
'Make this group a required group for its members
Set objProfile.IsMembershipRequired = true
...
```

## IsOrphan

Indicates whether this UNIX group profile is an orphan.

### Syntax

```
bool IsOrphan {get;}
```

### Property value

Returns `true` if the `GroupUnixProfile` object has no corresponding Active Directory group object, or `false` if the object has a corresponding Active Directory group object.

### Discussion

The UNIX group profile is an orphan if the corresponding Active Directory group object is missing.

### Exceptions

`IsOrphan` throws an `ApplicationException` if the group profile does not exist.

## Example

The following code sample illustrates using `IsOrphan` in a script:

```
...
'Identify the zone you want to work with
set objZone = cims.GetZone("ajax.org/UNIX/Zones/eur007")
'Check for orphan profiles
for each profile in objZone.GetGroupUnixProfiles
    If profile.IsOrphan then
        wScript.Echo profile.Name
    end if
end for
...
```

• • • • •

next  
...

## IsReadable

Indicates whether the group profile object in Active Directory is readable for the current user credentials.

### Syntax

```
bool IsReadable {get;}
```

### Property value

Returns `true` if the `GroupUnixProfile` object is readable, or `false` if the object is not readable.

### Discussion

This property returns a value of `true` if the user accessing the group profile object in Active Directory has sufficient permissions to read its properties.

### Example

The following code sample illustrates using `IsReadable` in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the Active Directory group
Set objGroup = cims.GetGroupByPath
("LDAP://CN=testers,CN=groups,DC=sierra,DC=com")
'Get the UNIX profile for the group in the zone
set objGroupUnixProfile = objGroupUnixProfiles.Find(objZone)
'Check whether the object is readable
if not objGroupUnixProfile.IsReadable then
    wscript.Echo "Denied read access. Exiting ...."
    wscript.Quit
else
    wscript.Echo "Read permission granted. Continuing ...."
    wscript.Echo "Group Profile GID: " & objGroupUnixProfile.GID
end if
...
```

## IsSFU

Indicates whether this group profile is an SFU zone profile.

• • • • •

## Syntax

```
bool IsSFU {get;}
```

## Property value

Returns true if the GroupUnixProfile object is an SFU zone profile.

## Discussion

See [Data storage for Centrify zones](#) for a discussion of SFU zones.

## IsWritable

Indicates whether the group profile object in Active Directory is writable for the current user's credentials.

## Syntax

```
bool IsWritable {get;}
```

## Property value

Returns true if the GroupUnixProfile object is writable, or false if the object is not writable.

## Discussion

This property returns a value of true if the user accessing the group profile object in Active Directory has sufficient permissions to change the group profile object's properties.

## Example

The following code sample illustrates using IsWritable in a script:

```
...
set objZone = cims.GetZoneByPath
("LDAP://CN=research,CN=zones,CN=centrify,CN=program
data,DC=sierra,DC=com")
'Identify the Active Directory group
set objGroup = cims.GetGroupByPath
("LDAP://CN=testers,CN=groups,DC=sierra,DC=com")
'Get the UNIX profile for the group in the zone
set objGroupUnixProfile = objGroupUnixProfiles.Find(objZone)
'Check whether the object is writable
if not objGroupUnixProfile.IsWritable then
```

• • • • •

```
wScript.Echo "Denied write access. Exiting ...."  
wScript.Quit  
else  
    wScript.Echo "Write permission granted. Continuing ...."  
    wScript.Echo "Group Profile GID: " & objGroupUnixProfile.GID  
end if  
...
```

## Members

Gets an existing list of members or sets a new list of members for the UNIX group profile associated with the specified local group.

### Syntax

```
string[] Members{get; set;}
```

### Property value

The members of a local group.

### Exceptions

Members throws an `InvalidOperationException` if the group you specify is not a local group.

## ProfileState

Gets the profile state of an existing local group or sets the profile of the specified local group.

### Syntax

```
GroupProfileState ProfileState{get; set;}
```

### Property value

The profile state of the specified local group.

### Exceptions

ProfileState throws an `InvalidOperationException` if the group you specify is not a local group.



## Name

Gets an existing name or sets a new name for the UNIX group profile associated with the specified Active Directory group in the specified zone.

## Syntax

```
string Name {get; set;}
```

## Property value

The UNIX group name for the UNIX profile in the zone.

## Example

The following code sample illustrates using Name in a script:

```
...
set objZone = cims.GetZoneByPath
("LDAP://CN=research,CN=zones,CN=centrify,CN=program
data,DC=sierra,DC=com")
'Identify the Active Directory group
Set objGroup = cims.GetGroupByPath
("LDAP://CN=managers,CN=groups,DC=sierra,DC=com")
'Get the UNIX profile for the group in the zone
set objGroupUnixProfile = objGroupUnixProfiles.Find(objZone)
wScript.Echo "Group Profile Name: " & objGroupUnixProfile.Name
...
```

## Type

Gets the type of the UNIX group profile.

## Syntax

```
GroupUnixProfileType Type {get;}
```

## Property value

Returns one of the following numeric values depending on how the UNIX group profile is stored:

- 0 indicates the group profile is a standard Centrify UNIX profile.
- 1 indicates the profile is a private group stored in a Private Groups container.

• • • • •

- 2 indicates the profile is a Centrify SFU profile.
- 3 indicates the profile is a local group.

## Discussion

The Private group type is only applicable to early versions of Centrify software. It is not a valid group profile type in version 4.0 and later.

## Example

The following code sample illustrates using Type in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data,DC=sierra,DC=com")
'Identify the Active Directory group
set objGroup = cims.GetGroupByPath
("LDAP://CN=escalation,CN=support,DC=sierra,DC=com")
'Get the UNIX profile for the group in the zone
set objGroupUnixProfile = objGroupUnixProfiles.Find(objZone)
select case objGroupUnixProfile.Type
case 0
    wscript.Echo "Standard group"
case 1
    wscript.Echo "Private group"
case 2
    wscript.Echo "SFU group"
end select
...
```

## UnixEnabled

Determines whether the UNIX information is enabled.

### Syntax

```
bool UnixEnabled {get; set;}
```

### Property value

Set true if the UNIX information is enabled.

## Example

For a code sample that uses the `UnixEnabled` property, see [AddUnixProfile](#).



## Zone

Gets the zone object for the current group unix profile.

### Syntax

```
IZone Zone {get;}
```

### Property value

The zone object for the UNIX group profile.

### Discussion

This property serves as a shortcut for retrieving data.

## GroupUnixProfiles

The `GroupUnixProfiles` class manages a collection of group profiles in a zone.

### Syntax

```
public interface IGroupUnixProfiles
```

### Discussion

The content of the collection of group profiles contained in the object depends on how the object was obtained:

- When you use `GetUserUnixProfiles`, the `GroupUnixProfiles` object returned enumerates all of the profiles defined for a specific Active Directory user across all zones in the current domain.
- When you use `GetGroupUnixProfiles`, the `GroupUnixProfiles` object returned enumerates all of the profiles defined for a specific Active Directory group in a specific zone.



## Methods

The `GroupUnixProfiles` class provides the following methods:

This method	Does this
<code>GetEnumerator</code>	Returns an enumeration of <code>GroupUnixProfile</code> objects.
<code>Refresh</code>	Reloads the <code>GroupUnixProfiles</code> object data from the data in Active Directory.

## Properties

The `GroupUnixProfiles` class provides the following properties:

This property	Does this
<code>Count</code>	Gets the total number of UNIX group profiles in the collection of <code>GroupUnixProfiles</code> for an Active Directory group.
<code>IsEmpty</code>	Indicates whether the <code>GroupUnixProfiles</code> object contains any UNIX group profiles.

## GetEnumerator

Returns an enumeration of `GroupUnixProfile` objects.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

An enumeration of `GroupUnixProfile` objects.

## Refresh

Reloads the `GroupUnixProfiles` object data from the data in Active Directory.

### Syntax

```
void Refresh()
```



## Discussion

This method refreshes the collection of group profiles in the cached object to ensure it is synchronized with the latest information in Active Directory.

## Count

Gets the total number of UNIX group profiles defined in the `GroupUnixProfiles` collection for an Active Directory group or zone.

## Syntax

```
int Count {get;}
```

## Property value

The number of UNIX group profiles in the `GroupUnixProfiles` collection.

## Discussion

This property enumerates all of the profiles in the collection before it returns the count value. If you only need to determine whether any profiles are defined, use the `IsEmpty` property for a faster response time.

## Example

The following code sample illustrates using `Count` in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data, DC=sierra, DC=com")
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data, DC=sierra, DC=com")
If objGroupUnixProfiles.IsEmpty then
    wscript.echo "No profiles defined"
Else
    wscript.echo objGroupUnixProfiles.Count & " profiles defined"
End if
...
```

## IsEmpty

Indicates whether the collection of UNIX group profiles is empty.

• • • • •

## Syntax

```
bool IsEmpty {get;}
```

## Property value

Returns `true` if there are no group profiles in the `GroupUnixProfiles` object, or `false` if there is at least one UNIX group profile in the object.

## Discussion

Unlike the `Count` property, the `IsEmpty` property does not query all of the profiles in the collection before it returns a value. If you only need to determine whether any profiles are defined, call this property for a faster response.

## Example

The following code sample illustrates using `IsEmpty` in a script:

```
...
set objZone = cims.GetZoneByPath("LDAP://CN=research,
CN=zones,CN=centrify,CN=program data, DC=sierra, DC=com")
If objGroupUnixProfiles.IsEmpty then
    wscript.echo "No profiles defined"
Else
    wscript.echo objGroupUnixProfiles.Count & " profiles defined"
End if
...
```

## HierarchicalGroup

The `HierarchicalGroup` class manages the UNIX group profile information of an Active Directory group in a hierarchical zone, as well as local groups.

## Syntax

```
public interface IHierarchicalGroup : IGroupUnixProfile
```

## Methods

The `HierarchicalGroup` class provides the following methods:



This method	Does this
<code>Commit</code>	Commits changes to the <code>GroupUnixProfile</code> object to Active Directory.  (Inherited from <code>GroupUnixProfile</code> .)
<code>Delete</code>	Marks the UNIX group profile object for deletion from Active Directory.  (Inherited from <code>GroupUnixProfile</code> .)
<code>GetComputer</code>	Returns the computer to which this group profile belongs.
<code>GetDirectoryEntry</code>	Returns the directory entry for a UNIX group profile from Active Directory.  (Inherited from <code>GroupUnixProfile</code> .)
<code>InheritFromParent</code>	Clears all property values so that all UNIX attributes for this user are inherited from the parent zone.
<code>Refresh</code>	Reloads the <code>GroupUnixProfile</code> object data from the data in Active Directory.  (Inherited from <code>GroupUnixProfile</code> .)
<code>ResolveEffectiveProfile</code>	Resolves the effective profile.
<code>Validate</code>	Validates data in the <code>GroupUnixProfile</code> object before the changes are committed to Active Directory.  (Inherited from <code>GroupUnixProfile</code> .)

## Properties

The `HierarchicalGroup` class provides the following properties:

This property	Does this
<code>ADSPath</code>	Gets the LDAP path to the UNIX group profile.  (Inherited from <code>GroupUnixProfile</code> .)
<code>Cims</code>	Gets the Cims data for the group profile.  (Inherited from <code>GroupUnixProfile</code> .)
<code>EffectiveGid</code>	Gets the effective GID of the group.
<code>EffectiveIsMembershipRequired</code>	Indicates whether members of this group can remove the group from their currently active set of groups (not applicable to local groups).
<code>EffectiveMembers</code>	Gets members of the local group (local groups only).



This property	Does this
<code>EffectiveName</code>	Gets the UNIX name of the group (not applicable to local groups).
<code>EffectiveProfileState</code>	Gets the profile state of the local group (local groups only).
<code>Group</code>	Gets the Active Directory group to which the <code>GroupUnixProfile</code> object belongs (not applicable to local groups).  (Inherited from <code>GroupUnixProfile</code> .)
<code>GroupID</code>	Gets or sets the numeric group identifier (GID) for the group profile.  (Inherited from <code>GroupUnixProfile</code> .)
<code>ID</code>	Gets the unique identifier for the UNIX group profile.  (Inherited from <code>GroupUnixProfile</code> .)
<code>IsEffectiveGidDefined</code>	Indicates whether there is an effective GID for this group.
<code>IsEffectiveIsMembershipRequiredDefined</code>	Indicates whether there is an effective membership requirement for this group (not applicable to local groups).
<code>IsEffectiveMembersDefined</code>	Indicates whether <code>EffectiveMembers</code> is defined for this group (local groups only).
<code>IsEffectiveNameDefined</code>	Indicates whether there is an effective name for this group.
<code>IsEffectiveProfileStateDefined</code>	Indicates whether there is an effective profile state defined for this group (local groups only).
<code>IsProfileStateDefined</code>	Determines whether the profile state is defined for this group (local groups only).
<code>IsForeign</code>	Indicates whether the UNIX profile for a group is in a different forest than its corresponding Active Directory group (not applicable to local groups).  (Inherited from <code>GroupUnixProfile</code> .)
<code>IsGidDefined</code>	Determines whether the GID is defined for this group.
<code>IsMembersDefined</code>	Determines whether <code>Members</code> is defined for this local group (local groups only).
<code>IsMembershipRequired</code>	Determines whether an Active Directory group is a required group (not applicable to local groups).



This property	Does this
	(Inherited from <a href="#">GroupUnixProfile</a> .)
<b>IsMembershipRequiredDefined</b>	Determines whether the membership requirement is defined for this group (not applicable to local groups).
<b>IsNameDefined</b>	Determines whether a name is defined for this group.
<b>IsOrphan</b>	Indicates whether this UNIX group profile is an orphan (not applicable to local groups). (Inherited from <a href="#">GroupUnixProfile</a> .)
<b>IsReadable</b>	Determines whether the Active Directory object is readable. (Inherited from <a href="#">GroupUnixProfile</a> .)
<b>IsSFU</b>	Indicates whether this UNIX group is an SFU zone profile (not applicable to local groups). (Inherited from <a href="#">GroupUnixProfile</a> .)
<b>IsWritable</b>	Determines whether the Active Directory object is writable. (Inherited from <a href="#">GroupUnixProfile</a> .)
<b>Members</b>	Gets an existing list of members or sets a new list of members for the UNIX group profile associated with the specified local group (local groups only). (Inherited from <a href="#">GroupUnixProfile</a> .)
<b>Name</b>	Gets or sets the group name of the UNIX group profile. (Inherited from <a href="#">GroupUnixProfile</a> .)
<b>ProfileState</b>	Gets or sets the profile state of a local group (local groups only). (Inherited from <a href="#">GroupUnixProfile</a> .)
<b>Type</b>	Gets the type of the UNIX group profile. (Inherited from <a href="#">GroupUnixProfile</a> .)



This property	Does this
<code>UnixEnabled</code>	Determines whether the UNIX information is enabled.  (Inherited from <code>GroupUnixProfile</code> .)
<code>Zone</code> <code>Zone</code>	Gets the zone associated with the UNIX group  (inherited from <code>GroupUnixProfile</code> )  Gets the zone to which this group profile belongs.

## GetComputer

Returns the computer to which this group profile belongs.

### Syntax

```
IHierarchicalZoneComputer GetComputer()
```

### Return value

Returns a hierarchical zone computer object specifying the computer to which this group profile belongs. Returns `null` if the group profile is not associated with a specific computer.

## InheritFromParent

This method clears all current-level property values so that all property values are inherited from ancestor zones or from defaults.

### Syntax

```
void InheritFromParent()
```

### Discussion

This method clears all current-level property values so that all property values are inherited from ancestor zones or from defaults. This is a convenience method that is equivalent to resetting all properties to `null`.



## ResolveEffectiveProfile

Resolves the effective profile for the group.

### Syntax

```
void ResolveEffectiveProfile()
```

### Discussion

This method resolves profiles of the group defined in the current zone, in parent zones, and in zone default values to determine the effective profile. If an error occurs, such as one of the parent zones not being accessible, the effective profile properties show the best-effort data retrieved before the error occurred.

## EffectiveGid

Gets the GID of the group.

### Syntax

```
long EffectiveGid {get;}
```

### Property value

The GID in the UNIX group profile.

### Exceptions

`EffectiveGid` throws an `InvalidOperationException` if the UNIX group profile does not include a GID.

## EffectiveMembers

Gets the members of the local group.

### Syntax

```
string[] EffectiveMembers {get;}
```



## Property value

The members of the group.

## Exceptions

`EffectiveGid` throws an `InvalidOperationException` if the UNIX group profile does not have members defined, or if this is not a local group profile and you attempt to set or get this property.

## EffectiveIsMembershipRequired

Indicates whether members of this group can remove the group from their currently active set of groups.

## Syntax

```
bool EffectiveIsMembershipRequired {get;}
```

## Property value

Returns `true` if you can use the `adsetgroups` command to remove this group from your currently active set of groups.

## Discussion

On most UNIX systems, a user can be a member of only a limited number of groups at one time. Because of this limitation, it is useful to be able to change a user's group membership by adding and removing groups when necessary.

You can use the `adsetgroups` command to manage the set of Active Directory groups that are available to a UNIX account. You also have the option to specify that membership in a specific group is required in a zone.

If you specify that a group is required, users who are members of the group cannot remove that group from their currently active set of groups. In that case, the `EffectiveIsMembershipRequired` property returns `false`.

## Exceptions

`IsEffectiveMembershipRequired` throw an `InvalidOperationException` if there is only a partial profile.

• • • • •

## EffectiveName

Gets the UNIX name of the group.

### Syntax

```
string EffectiveName {get;}
```

### Property value

The group name in the UNIX group profile.

## EffectiveProfileState

Gets the profile state of the local group.

### Syntax

```
GroupProfileState EffectiveProfileState {get;}
```

### Property value

The profile state of the local group.

### Exceptions

`EffectiveProfileState` throws an `InvalidOperationException` if the UNIX group profile does not have a profile state defined, or if this is not a local group profile and you attempt to get this property.

## IsEffectiveGidDefined

Indicates whether there is an effective GID for this group.

### Syntax

```
bool IsEffectiveGidDefined {get;}
```

### Property value

Returns `true` if there is a UNIX group identifier (GID) in the UNIX group profile.



## Discussion

If the `ResolveEffectiveProfile` method has not been called, the value is resolved the first time this property is accessed.

## `IsEffectiveIsMembershipRequiredDefined`

Indicates whether there is an effective membership requirement for this group.

## Syntax

```
bool IsEffectiveIsMembershipRequiredDefined {get;}
```

## Property value

Returns `true` if there is an effective membership requirement for this group.

## Discussion

If the `ResolveEffectiveProfile` method has not been called, the value is resolved the first time this property is accessed.

## `IsEffectiveMembersDefined`

Indicates whether there is an effective members requirement for this local group.

## Syntax

```
bool IsEffectiveMembersDefined {get;}
```

## Property value

Returns `true` if there is an effective members requirement for this group.

## Discussion

If the `ResolveEffectiveProfile` method has not been called, the value is resolved the first time this property is accessed.



## Exceptions

`IsEffectiveMembersDefined` throws an `InvalidOperationException` if this is not a local group profile and you attempt to get this property.

This property is only applicable to local groups.

## IsEffectiveNameDefined

Indicates whether there is an effective name for this group.

### Syntax

```
bool IsEffectiveNameDefined {get;}
```

### Property value

Returns true if there is an effective name for this group.

### Discussion

If the `ResolveEffectiveProfile` method has not been called, the value is resolved the first time this property is accessed.

## IsEffectiveProfileStateDefined

Indicates whether there is an effective profile state for this local group.

### Syntax

```
bool IsEffectiveProfileStateDefined {get;}
```

### Property value

Returns true if there is an effective profile state for this group.

### Discussion

If the `ResolveEffectiveProfile` method has not been called, the value is resolved the first time this property is accessed.

This property is only applicable to a local group profile.



## Exceptions

`IsEffectiveProfileStateDefined` throws an `InvalidOperationException` if this is not a local group profile and you attempt to get this property.

## IsGidDefined

Determines whether there is a GID defined for this group.

### Syntax

```
bool IsGidDefined {get; set;}
```

### Property value

Returns `true` if there is a GID defined for this group. Set this property `false` to clear the GID.

## Exceptions

`IsGidDefined` throws an `InvalidOperationException` if the GID has not been defined and you attempt to set this property `true`.

## IsMembersDefined

Indicates whether `Members` is defined for this local group.

### Syntax

```
bool IsMembersDefined {get; set;}
```

### Property value

Returns `true` if `Members` is defined for this group. Set this property `false` to clear `Members`.

## Exceptions

`IsMembers` throws an `InvalidOperationException` if:

- `Members` has not been defined and you attempt to set this property `true`.
- if this is not a local group and you attempt to set or get this property.



## IsMembershipRequiredDefined

Determines whether the membership requirement is defined for this group.

### Syntax

```
bool IsMembershipRequiredDefined {get; set;}
```

### Property value

Returns `true` if the membership requirement is defined for this group. Set this property `false` to clear the `IsMembershipRequired` flag.

### Exceptions

`IsMembershipRequiredDefined` throws an `InvalidOperationException` if the `IsMembershipRequired` flag has not been defined and you attempt to set this property `true`.

## IsNameDefined

Determines whether the name is defined for this group.

### Syntax

```
bool IsNameDefined {get; set;}
```

### Property value

Returns `true` if the name is defined for this group. Set this property `false` to clear the name.

### Exceptions

`IsNameDefined` throws an `InvalidOperationException` if the name has not been defined and you attempt to set this property `true`.

## IsProfileStateDefined

Indicates whether there is a profile state defined for this local group.



## Syntax

```
bool IsProfileStateDefined {get;}
```

## Property value

Returns true if there is an a profile state defined for this group.

## Discussion

Setting this property to false will clear **ProfileState**.

## Exceptions

`IsProfileStateDefined` throws an `InvalidOperationException` if:

- **ProfileState** is not defined and you attempt to set this property to true.
- this is not a local group profile and you attempt to set or get this property.

## Zone

Gets the zone to which this group profile belongs.

## Syntax

```
IHierarchicalZone Zone {get;}
```

## Property value

The zone to which this group profile belongs; null if this is a computer-specific profile.

## HierarchicalUser

The `HierarchicalUser` class manages the UNIX user profile information of an Active Directory user in a hierarchical zone.

## Syntax

```
public interface IHierarchicalUser : IUserUnixProfile
```



## Discussion

In hierarchical zones, both identity (profile data) and access (authorization data) are inherited, such that a user's effective identity or access are determined by all the profile data and all the access data at all levels of the hierarchy.

Profile data can be defined at any level: parent, child, or computer. It is possible to define a partial profile at any level—that is, leave one or more of the NSS fields blank. Although a complete profile is required to have access to a machine, a profile in a child zone can complete the missing fields from the parent zone. In the case of conflict, profile definitions in a child zone override the definition in the parent zone and computer-level definitions override all zone-level definitions.

On the other hand, role assignments do not override each other. Rather, they accumulate, such that a user's potential rights include all the rights granted by all the role assignments in the access tree. These are potential rights because rights granted to a user by a role assignment are effective only if the user has a complete profile defined for a zone.

In other words, when a computer joins a zone, the profile tree determines a pool of potential users, the access tree determines a different set of users with rights, and where the two intersect is the set of effective users.

See the `WindowsUser` class for a user's Windows profile.

## Methods

The `HierarchicalUser` class provides the following methods:

This method	Does this
<code>AddUserRoleAssignment</code>	Returns a new user role assignment.
<code>Commit</code>	Commits changes to the <code>userUnixProfile</code> object to Active Directory. (Inherited from <code>UserUnixProfile</code> .)
<code>Delete</code>	Marks the UNIX user profile object for deletion from Active Directory. (Inherited from <code>UserUnixProfile</code> .)
<code>GetComputer</code>	Returns the computer to which this user profile belongs.



This method	Does this
<code>GetDirectoryEntry</code>	Returns the directory entry for a UNIX user profile from Active Directory.  (Inherited from <code>UserUnixProfile</code> .)
<code>GetEffectiveUserRoleAssignments</code>	Returns the effective user role assignments.
<code>GetPrimaryGroup</code>	Returns the UNIX profile of the primary group of the user.  (Inherited from <code>UserUnixProfile</code> .)
<code>GetUserRoleAssignment</code>	Returns a user role assignment for this UNIX user.
<code>GetUserRoleAssignments</code>	Returns all the user role assignments for this UNIX user.
<code>InheritFromParent</code>	Clears all property values so that all UNIX attributes for this user are inherited from the parent zone.
<code>Refresh</code>	Reloads the <code>userUnixProfile</code> object data from the data in Active Directory.  (Inherited from <code>UserUnixProfile</code> .)
<code>ResolveEffectiveProfile</code>	Resolves the effective profile to be used when the user logs on to the computer.
<code>ResolveEffectiveRoles</code>	Resolves the effective roles for this user.
<code>Validate</code>	Validates data in the <code>userUnixProfile</code> object before the changes are committed to Active Directory.  (Inherited from <code>UserUnixProfile</code> .)

## Properties

The `HierarchicalUser` class provides the following properties:

This property	Does this
<code>ADsPath</code>	Gets the LDAP path to the UNIX user profile.  (Inherited from <code>UserUnixProfile</code> .)
<code>Cims</code>	Gets the Cims data for the user profile.  (Inherited from <code>UserUnixProfile</code> .)
<code>EffectiveGecos</code>	Gets the contents of the effective GECOS field of the user profile.
<code>EffectiveGecosZone</code>	Gets the hierarchical zone of the effective GECOS.
<code>EffectiveHomeDirectory</code>	Gets the effective home directory of the user.



<b>This property</b>	<b>Does this</b>
<b>EffectiveHomeDirectoryZone</b>	Gets the zone of the user's home directory.
<b>EffectiveIsUseAutoPrivateGroup</b>	Indicates whether this user uses an auto private group (not applicable to local user profiles).
<b>EffectiveName</b>	Gets the user's effective logon name.
<b>EffectiveNameZone</b>	Gets the zone of the user's effective UNIX name.
<b>EffectivePrimaryGroup</b>	Gets the effective primary group GID of the user.
<b>EffectivePrimaryGroupZone</b>	Gets the zone of the primary group GID.
<b>EffectiveProfileState</b>	Gets the effective profile state of the local user (local user profiles only).
<b>EffectiveProfileStateZone</b>	Gets the zone which defines the effective profile state
<b>EffectiveShell</b>	Gets the effective logon shell of the user.
<b>EffectiveShellZone</b>	Gets the zone of the effective logon shell.
<b>EffectiveUid</b>	Gets the effective UID of the user.
<b>EffectiveUidZone</b>	Gets the zone of the user's effective UID.
<b>Gecos</b>	Gets or sets the contents of the GECOS field explicitly set in the user profile of the current zone.
<b>HomeDirectory</b>	Gets or sets the home directory of the user. (Inherited from <a href="#">UserUnixProfile</a> .)
<b>ID</b>	Gets the unique identifier for the UNIX user profile. (Inherited from <a href="#">UserUnixProfile</a> .)
<b>IsEffectiveGecosDefined</b>	Indicates whether there is an effective GECOS for this user.
<b>IsEffectiveHomeDirectoryDefined</b>	Indicates whether there is an effective home directory defined for this user.
<b>IsEffectiveNameDefined</b>	Indicates whether there is an effective name for this user.
<b>IsEffectivePrimaryGroupDefined</b>	Indicates whether a primary group is defined for this user.
<b>IsEffectiveProfileStateDefined</b>	Indicates whether there is an effective profile state for this local user (local user profiles only).
<b>IsEffectiveShellDefined</b>	Indicates whether there is an effective shell defined for this user.



This property	Does this
<code>IsEffectiveUidDefined</code>	Indicates whether the user has an effective UID.
<code>IsEffectiveUseAutoPrivateGroupDefined</code>	Indicates whether the auto private group flag is defined for this user (not applicable to local user profiles).
<code>IsForeign</code>	Indicates whether the UNIX profile for a user is in a different forest than its corresponding Active Directory user (not applicable to local user profiles).  (Inherited from <code>UserUnixProfile</code> .)
<code>IsGecosDefined</code>	Determines whether the GECOS is defined in this profile.
<code>IsHomeDirectoryDefined</code>	Determines whether the home directory is defined in this profile.
<code>IsNameDefined</code>	Determines whether a name is defined in this profile.
<code>IsOrphan</code>	Indicates whether this UNIX user profile is an orphan (not applicable to local user profiles).  (Inherited from <code>UserUnixProfile</code> .)
<code>IsPrimaryGroupDefined</code>	Determines whether there is a GID defined for this user in this zone.
<code>IsProfileStateDefined</code>	Gets or sets whether the profile state is defined in this local user profile (local user profiles only).
<code>IsReadable</code>	Determines whether the Active Directory object is readable.  (Inherited from <code>UserUnixProfile</code> .)
<code>IsSecondary</code>	Indicates whether this is a secondary profile (not applicable to local user profiles).
<code>IsSFU</code>	Indicates whether this user object uses the Microsoft Services for UNIX (SFU) schema extension (not applicable to local user profiles).  (Inherited from <code>UserUnixProfile</code> .)
<code>IsShellDefined</code>	Determines whether the shell is defined in this profile.
<code>IsUidDefined</code>	Determines whether the ID is defined in this profile.
<code>IsUseAutoPrivateGroup</code>	Determines whether this user uses auto private groups (not applicable to local user



This property	Does this
	profiles).
<b>IsUseAutoPrivateGroupDefined</b>	Determines whether the auto private group flag is defined (not applicable to local user profiles).
<b>IsWritable</b>	Determines whether the Active Directory object is writable. (Inherited from <a href="#">UserUnixProfile</a> .)
<b>Name</b>	Gets or sets the user name of the UNIX user profile. (Inherited from <a href="#">UserUnixProfile</a> .)
<b>PrimaryGroup</b>	Gets or sets the GID of the user's primary group. (Inherited from <a href="#">UserUnixProfile</a> .)
<b>ProfileState</b>	Gets or sets the profile state of a local user profile (local user profiles only). (Inherited from <a href="#">UserUnixProfile</a> )
<b>Shell</b>	Gets or sets the user's default shell. (Inherited from <a href="#">UserUnixProfile</a> .)
<b>Type</b>	Gets the type of the UNIX user profile. (Inherited from <a href="#">UserUnixProfile</a> .)
<b>UnixEnabled</b>	Determines whether the UNIX information is enabled. (Inherited from <a href="#">UserUnixProfile</a> .)
<b>User</b>	Gets the user to whom this UNIX profile belongs (not applicable to local user profiles). (Inherited from <a href="#">UserUnixProfile</a> .)
<b>UserId</b>	Gets or sets the user identifier (UID) for the user profile. (Inherited from <a href="#">UserUnixProfile</a> .)
<b>Zone</b>	Gets the zone associated with the UNIX user (inherited from <a href="#">UserUnixProfile</a> ) Gets the zone to which this user profile belongs.



## AddUserRoleAssignment

Adds a user role assignment to the user profile.

### Syntax

```
IRoleAssignment AddUserRoleAssignment()
```

### Return value

An empty user role assignment object. This role assignment is not stored in Active Directory until you call the `RoleAssignment.Commit` method.

### Discussion

This object is not saved to Active Directory until you set at least one property value and call the `Commit` method.

### Example

The following code sample illustrates using `AddUserRoleAssignment` in a script:

```
...
IHierarchicalUser objUserUnixProfile = (IHierarchicalUser)
objZone.GetUserUnixProfile(objUser);
if (objUserUnixProfile == null)
{
    //New user for the zone
    objUserUnixProfile = objZone.AddUserPartialProfile(strUser);
}
IRole objRole = objZone.GetRole(strRole);
if (objRole == null)
{
    Console.WriteLine("Role " + strRole + " does not exist.");
    return;
}
IRoleAssignment asg = objUserUnixProfile.GetUserRoleAssignment
(objRole);
if (asg != null)
{
    Console.WriteLine("Assignment already exist.");
    return;
}
else
{
    // assigning role to user
    asg = objUserUnixProfile.AddUserRoleAssignment();
    asg.Role = objZone.GetRole(strRole);
    asg.Commit();
    Console.WriteLine("Role " + strRole + " was successfully
```

• • • • •

```
assigned to " + strUser + ".");  
}  
...
```

## GetComputer

Returns the computer to which this user profile belongs.

### Syntax

```
IHierarchicalZoneComputer GetComputer ()
```

### Return value

Returns a hierarchical zone computer object specifying the computer to which this user profile belongs. Returns null if the user profile is not associated with a specific computer.

## GetEffectiveUserRoleAssignments

Returns an enumeration of the effective user role assignments.

### Syntax

```
IRoleAssignments GetEffectiveUserRoleAssignments()
```

### Return value

An enumeration of the effective user role assignments for this user.

### Discussion

The collection of effective role assignments is a combination of all the role assignments for this user in this zone and all parent zones. See the [HierarchicalUser](#) class for a more complete discussion.

## GetUserRoleAssignment

Returns the role assignment for a specific role for this user.



## Syntax

```
IRoleAssignment GetUserRoleAssignment(IRole role)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
role	The role for which you want the role assignment.

## Return value

The role assignment that associates this user with the specified role. Returns `null` if none exists.

## Exceptions

`GetUserRoleAssignment` throws an `ArgumentNullException` if you pass `null` for the `role` parameter.

## Example

The following code sample illustrates using `GetUserRoleAssignment` in a script:

```
...
// Create a CIMS object to interact with AD'
ICims cims = new Cims();
// Note the lack of the cims.connect function.'
// By default, this application will use the connection to the
domain controller
// and existing credentials from the computer already logged in.
IHierarchicalZone objZone =
cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
IUser objUser = cims.GetUserByPath(strUser);
if (objUser == null)
{
    Console.WriteLine("User " + strUser + " does not exist.");
    return;
}
IHierarchicalUser objUnixUser = objZone.GetUserUnixProfile
(objUser) as IHierarchicalUser;
if (objUnixUser == null)
{
    objUnixUser = objZone.AddUserPartialProfile(strUser);
}
IRole objRole = objZone.GetRole(strRoleName);
if (objRole == null)
{
```

• • • • •

```
        Console.WriteLine("Role " + strRoleName + " does not  
exist.");  
        return;  
    }  
    IRoleAssignment objAsg = objUnixUser.GetUserRoleAssignment  
(objRole);  
    if (objAsg == null)  
    {  
        Console.WriteLine("Role assginment does not exist.");  
        return;  
    }  
    else  
    {  
        objAsg.Delete();  
        Console.WriteLine("Role " + strRoleName + " was successfully  
removed from user "  
            + strUser);  
    }  
    ...
```

## GetUserRoleAssignments

Returns an enumeration of the role assignments for this UNIX user.

### Syntax

```
IRoleAssignments GetUserRoleAssignments()
```

### Return value

An enumeration of the role assignments for this user in this zone.

### Discussion

Call the `GetEffectiveUserRoleAssignments` method to get the effective collection of role assignments, including those defined for this user in parent zones.

## InheritFromParent

Clears all property values so that all UNIX attributes for this user are inherited from the parent zone.

### Syntax

```
void InheritFromParent()
```



## Discussion

This method clears all current-level property values so that all property values are inherited from ancestor zones or from defaults. This is a convenience method that is equivalent to resetting all properties to `null`.

## ResolveEffectiveProfile

Resolves the profile for the user that is effective when the user logs on to the computer.

## Syntax

```
void ResolveEffectiveProfile()
```

## Discussion

This method resolves the profiles of the user in the current zone and parent zones, plus zone default values (if any), to determine effective profile values. If an error occurs, such as one of the parent zones not being accessible, the effective profile properties show the best-effort data retrieved before the error occurred.

You must call this method before calling any of the properties that return effective profile values.

## ResolveEffectiveRoles

Resolves the effective roles for the user.

## Syntax

```
void ResolveEffectiveRoles(IHierarchicalZone zone)
void ResolveEffectiveRoles(IHierarchicalZoneComputer computer)
```

## Parameters

Specify one of the following parameters when using this method.

Parameter	Description
<code>zone</code>	The zone for which you want the group and user role assignments.
<code>computer</code>	The computer for which you want the group and user role assignments.



## Discussion

This method resolves the groups and roles of the user in the specified zone or computer and parent zones. If you specify a zone, the method ignores computer-level roles and groups. If you specify a computer, the method considers only roles and groups defined for that computer. For a discussion of roles in hierarchical zones, see the [HierarchicalUser](#) class.

## EffectiveGecos

Gets the effective GECOS field of the user profile.

### Syntax

```
string EffectiveGecos {get;}
```

### Property value

The contents of the GECOS field of the effective profile for this user.

## Discussion

You must call the [ResolveEffectiveProfile](#) method before calling this property. If you don't do so, this property returns the unresolved value at the current hierarchical level, or null if there is none.

## EffectiveGecosZone

Gets the zone in which the effective GECOS field is defined.

### Syntax

```
IHierarchicalZone EffectiveGecosZone {get;}
```

### Property value

The lowest-level hierarchical zone where the GECOS field is defined. This value overrides any definitions in higher-level zones.



## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns `null`.

Call the `Gecos` method to get or set the GECOS field for the current zone.

## EffectiveHomeDirectory

Gets the effective home directory of the user.

### Syntax

```
string EffectiveHomeDirectory {get;}
```

### Property value

The contents of the home directory field of the effective profile for this user.

## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns the unresolved value at the current hierarchical level, or `null` if there is none.

## EffectiveHomeDirectoryZone

Gets the zone in which the effective home directory of the user is defined.

### Syntax

```
IHierarchicalZone EffectiveHomeDirectoryZone {get;}
```

### Property value

The lowest-level hierarchical zone where the home directory field is defined. This value overrides any definitions in higher-level zones.

## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns `null`.



## EffectiveIsUseAutoPrivateGroup

Indicates whether the effective user profile enables auto private groups.

### Syntax

```
bool EffectiveIsUseAutoPrivateGroup {get;}
```

### Property value

Returns true if the effective profile for this user enables auto private groups.

### Discussion

Auto private group sets the user's UNIX profile name as the group name and the user's UID as the group GID.

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns the unresolved value at the current hierarchical level, or null if there is none.

## EffectiveName

Gets the user's effective logon name.

### Syntax

```
string EffectiveName {get;}
```

### Property value

The contents of the logon name field of the effective profile for this user.

### Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns the unresolved value at the current hierarchical level, or null if there is none.

## EffectiveNameZone

Gets the zone in which the effective logon name of the user is defined.



## Syntax

```
IHierarchicalZone EffectiveNameZone {get;}
```

## Property value

The lowest-level hierarchical zone where the logon name field is defined. This value overrides any definitions in higher-level zones.

## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns `null`.

## EffectivePrimaryGroup

Gets the GUID of the effective primary group from the user profile.

## Syntax

```
long EffectivePrimaryGroup {get;}
```

## Property value

The contents of the primary group field of the effective profile for this user.

## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns the unresolved value at the current hierarchical level, or `null` if there is none.

## EffectiveProfileState

Gets the effective profile state of the local user.

## Syntax

```
UserProfileState EffectiveProfileState{get;}
```

## Property value

The contents of the profile state field of the effective profile for this local user.



## Discussion

If `ResolveEffectiveProfile()` has not been called, this property will return either `null`, or the explicit value.

## Exceptions

`EffectiveProfileState` throws an `InvalidOperationException` if this is not a local user profile and you attempt to get this property.

## EffectiveProfileStateZone

Gets the zone which defines the effective profile state.

## Syntax

```
IHierarchicalZone EffectiveProfileStateZone{get;}
```

## Property value

The lowest-level hierarchical zone where the profile state field is defined. This value overrides any definitions in higher-level zones.

## Discussion

If `ResolveEffectiveProfile()` has not been called, this property will always return `null`.

## Exceptions

`EffectiveProfileStateZone` throws an `InvalidOperationException` if this is not a local user profile and you attempt to get this property.

## EffectivePrimaryGroupZone

Gets the zone in which the GUID of the effective primary group of the user is defined.



## Syntax

```
IHierarchicalZone EffectivePrimaryGroupZone {get;}
```

## Property value

The lowest-level hierarchical zone where the primary group field is defined. This value overrides any definitions in higher-level zones.

## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns `null`.

## EffectiveShell

Gets the user's effective logon shell.

## Syntax

```
string EffectiveShell {get;}
```

## Property value

The contents of the logon shell field of the effective profile for this user.

## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns the unresolved value at the current hierarchical level, or `null` if there is none.

## EffectiveShellZone

Gets the zone in which the user's effective logon shell is defined.

## Syntax

```
IHierarchicalZone EffectiveShellZone {get;}
```



## Property value

The lowest-level hierarchical zone where the logon shell field is defined. This value overrides any definitions in higher-level zones.

## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns `null`.

## EffectiveUid

Gets the user's effective UID.

## Syntax

```
long EffectiveUID {get;}
```

## Property value

The contents of the UID field of the user's effective profile.

## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns the unresolved value at the current hierarchical level, or `null` if there is none.

## EffectiveUidZone

Gets the zone in which the user's effective UID is defined.

## Syntax

```
IHierarchicalZone EffectiveUIDZone {get;}
```

## Property value

The lowest-level hierarchical zone where the UID field is defined. This value overrides any definitions in higher-level zones.

• • • • •

## Discussion

You must call the `ResolveEffectiveProfile` method before calling this property. If you don't do so, this property returns `null`.

## Gecos

Gets or sets the GECOS field of the user profile in the current zone.

## Syntax

```
string Gecos {get; set;}
```

## Property value

The contents of the GECOS field.

## Discussion

Call the `EffectiveGecos` method to get the effective GECOS for this zone.

## IsEffectiveGecosDefined

Indicates whether there is an effective value for the GECOS field for this user.

## Syntax

```
bool IsEffectiveGecosDefined {get;}
```

## Property value

Returns `true` if an effective value for the GECOS field exists for this user.

## Discussion

See the discussion of the `ResolveEffectiveProfile` method.

## IsEffectiveHomeDirectoryDefined

Indicates whether there is an effective home directory for this user.

• • • • •

### Syntax

```
bool IsEffectiveHomeDirectoryDefined {get;}
```

### Property value

Returns true if an effective home directory exists for this user.

### Discussion

See the discussion of the `ResolveEffectiveProfile` method.

## IsEffectiveNameDefined

Indicates whether there is an effective logon name for this user.

### Syntax

```
bool IsEffectiveNameDefined {get;}
```

### Property value

Returns true if an effective name exists for this user.

### Discussion

See the discussion of the `ResolveEffectiveProfile` method.

## IsEffectivePrimaryGroupDefined

Indicates whether there is an effective GID for this user.

### Syntax

```
bool IsEffectivePrimaryGroupDefined {get;}
```

### Property value

Returns true if an effective GID exists for this user.

### Discussion

See the discussion of the `ResolveEffectiveProfile` method.



## IsEffectiveProfileStateDefined

Indicates whether there is an effective profile state for this local user.

### Syntax

```
bool IsEffectiveProfileStateDefined {get;}
```

### Property value

Returns true if an effective profile state exists for this local user.

### Exceptions

`IsEffectiveProfileStateDefined` throws an `InvalidOperationException` if this is not a local user profile and you attempt to get this property.

## IsEffectiveShellDefined

Indicates whether there is an effective logon shell for this user.

### Syntax

```
bool IsEffectiveShellDefined {get;}
```

### Property value

Returns true if an effective logon shell exists for this user.

### Discussion

See the discussion of the `ResolveEffectiveProfile` method.

## IsEffectiveUidDefined

Indicates whether there is an effective UID for this user.

### Syntax

```
bool IsEffectiveUidDefined {get;}
```



## Property value

Returns `true` if there is an effective UNIX user identifier (UID) for this user.

## Discussion

See the discussion of the `ResolveEffectiveProfile` method.

## IsEffectiveUseAutoPrivateGroupDefined

Indicates whether there is an effective auto private group flag setting for this user.

## Syntax

```
bool IsEffectiveUseAutoPrivateGroupDefined {get;}
```

## Property value

Returns `true` if there is an effective auto private group flag for this user.

## Discussion

When auto private groups are enabled, the user's UNIX profile name is automatically used as the group name and the user's UID is used as the GID.

See the discussion of the `ResolveEffectiveProfile` method.

## IsGecosDefined

Determines whether there is a GECOS field defined for this user in this zone.

## Syntax

```
bool IsGecosDefined {get; set;}
```

## Property value

Returns `true` if there is a GECOS field defined for this user. Set this property `false` to clear the GECOS field.



## Exceptions

`IsGecosDefined` throws an `InvalidOperationException` if the GECOS field has not been defined and you attempt to set this property `true`.

## IsHomeDirectoryDefined

Determines whether there is a home directory defined for this user in this zone.

### Syntax

```
bool IsHomeDirectoryDefined {get; set;}
```

### Property value

Returns `true` if there is a home directory defined for this user. Set this property `false` to clear the home directory.

## Exceptions

`IsHomeDirectoryDefined` throws an `InvalidOperationException` if the home directory has not been defined and you attempt to set this property `true`.

## IsNameDefined

Determines whether there is a logon name defined for this user in this zone.

### Syntax

```
bool IsNameDefined {get; set;}
```

### Property value

Returns `true` if there is a logon name defined for this user. Set this property `false` to clear the logon name.

## Exceptions

`IsNameDefined` throws an `InvalidOperationException` if the name has not been defined and you attempt to set this property `true`.



## IsProfileStateDefined

Determines whether the profile state is defined for this local user profile.

### Syntax

```
bool IsProfileStateDefined {get; set;}
```

### Property value

Returns `true` if there is a profile state defined for this user. Set this property `false` to clear the profile state.

### Exceptions

`IsProfileStateDefined` throws an `InvalidOperationException` if:

- The profile state has not been defined and you attempt to set this property to `true`.
- This is not a local user profile and you attempt to set or get this property.

## IsPrimaryGroupDefined

Determines whether there is a primary GID defined for this user in this zone.

### Syntax

```
bool IsPrimaryGroupDefined {get; set;}
```

### Property value

Returns `true` if there is a primary GID defined for this user. Set this property `false` to clear the GID.

### Discussion

The user's primary group identifier (GID) can be associated with an Active Directory group or be a separate "dedicated-user" group that is only used in the UNIX operating environment. This property indicates whether a group profile for that GID has been defined in the zone.



## Exceptions

`IsPrimaryGroupDefined` throws an `InvalidOperationException` if the primary GID has not been defined and you attempt to set this property `true`.

## IsSecondary

Indicates whether the profile in this zone is a secondary profile.

## Syntax

```
bool IsSecondary {get;}
```

## Property value

Returns `true` if this is a secondary profile. Returns `false` if this is a primary profile.

## IsShellDefined

Determines whether there is a logon shell defined for this user in this zone.

## Syntax

```
bool IsShellDefined {get; set;}
```

## Property value

Returns `true` if there is a logon shell defined for this user. Set this property `false` to clear the shell.

## Exceptions

`IsShellDefined` throws an `InvalidOperationException` if the logon shell has not been defined and you attempt to set this property `true`.

## IsUidDefined

Determines whether there is a UID defined for this user in this zone.



## Syntax

```
bool IsUidDefined {get; set;}
```

## Property value

Returns `true` if there is a UID defined for this user. Set this property `false` to clear the UID.

## Exceptions

`IsUidDefined` throws an `InvalidOperationException` if the UID has not been defined and you attempt to set this property `true`.

## IsUseAutoPrivateGroup

Determines whether the user uses auto private groups.

## Syntax

```
bool IsUseAutoPrivateGroup {get; set;}
```

## Property value

Returns `true` if this user uses an auto private group.

## Discussion

When auto private groups are enabled, the user's UNIX profile name is automatically used as the group name and the user's UID is used as the GID.

## IsUseAutoPrivateGroupDefined

Determines whether the auto private group flag is defined for this user in this zone.

## Syntax

```
bool IsUseAutoPrivateGroupDefined {get; set;}
```



## Property value

Returns `true` if the auto private group flag is defined for this user. Set this property `false` to remove the flag definition from the profile.

## Discussion

When auto private groups are enabled, the user's UNIX profile name is automatically used as the group name and the user's UID is used as the GID.

## Exceptions

`IsUseAutoPrivateGroupDefined` throws an `InvalidOperationException` if the auto private group flag has not been defined and you attempt to set this property `true`.

## Zone

Gets the zone to which this user profile belongs.

## Syntax

```
IHierarchicalZone Zone {get;}
```

## Property value

The zone to which this user profile belongs; `null` if this is a computer-specific profile.

## HierarchicalZone

The `HierarchicalZone` class represents a hierarchical zone.

## Syntax

```
public interface IHierarchicalZone : IZone
```



## Discussion

The `HierarchicalZone` class inherits many methods and properties from the `Zone` class, but adds support for partial profiles and inheritable roles. Under hierarchical zones, both identity (profile data) and access (authorization data) are inherited, such that a user's effective identity or access are determined by all the profile data and all the access data at all levels of the hierarchy.

See [HierarchicalUser](#) for a discussion of profile and access inheritance.

## Methods

The `HierarchicalZone` class provides the following methods:

This method	Does this
<a href="#">AddAccessGroup</a>	Adds an empty role assignment to a group
<a href="#">AddComputerRole</a>	Creates a computer role under this zone.
<a href="#">AddGroupPartialProfile</a>	Adds a partial profile for a specified group.
<a href="#">AddLocalGroupPartialProfile</a>	Adds a partial profile for a specified local group.
<a href="#">AddLocalUserPartialProfile</a>	Adds a partial profile for a specified local user.
<a href="#">AddMitUser</a>	Adds an MIT Kerberos realm trusted user to this zone. (Inherited from <a href="#">Zone</a> .)
<a href="#">AddRoleAssignment</a>	Adds an empty role assignment.
<a href="#">AddUserPartialProfile</a>	Adds a partial profile for a specified user.
<a href="#">Commit</a>	Commits changes to the group object to Active Directory. (Inherited from <a href="#">Zone</a> .)
<a href="#">CreateCommand</a>	Creates a command right for the zone.
<a href="#">CreateImportPendingGroup</a>	Creates a pending imported group in this zone. (Inherited from <a href="#">Zone</a> .)
<a href="#">CreateImportPendingUser</a>	Creates a pending imported user in this zone. (Inherited from <a href="#">Zone</a> .)
<a href="#">CreateNetworkAccess</a>	Creates a network application access right.
<a href="#">CreatePamAccess</a>	Creates a PAM application access right.
<a href="#">CreateRole</a>	Creates a role in the zone.
<a href="#">CreateSshRight</a>	Creates an SSH application access right.



<b>This method</b>	<b>Does this</b>
<code>CreateWindowsApplication</code>	Creates a Windows application access right.
<code>CreateWindowsDesktop</code>	Creates a Windows Desktop access right.
<code>Delete</code>	Marks the zone for deletion from Active Directory. (Inherited from <code>Zone</code> .)
<code>GeneratePredefinedRights</code>	Generates predefined SSH and PAM rights in this zone.
<code>GeneratePredefinedRoles</code>	Generates predefined user roles in this zone.
<code>GetAccessGroup</code>	Returns a group assigned to this zone given a role for the group.
<code>GetAccessGroups</code>	Returns an enumeration of groups in the zone.
<code>GetChildZones</code>	Returns an enumeration of this zone's child zones.
<code>GetCommand</code>	Returns the privileged command right with a specific name or GUID.
<code>GetCommands</code>	Returns an enumeration of all the privileged command rights in the zone.
<code>GetComputerByDN</code>	Returns the computer profile in the zone given the distinguished name of the profile. (Inherited from <code>Zone</code> .)
<code>GetComputerRole</code>	Returns a specific computer role under this zone.
<code>GetComputerRoles</code>	Returns an enumeration of all the computer roles under this zone.
<code>GetComputers</code>	Returns an enumeration of all the computers in the zone. (Inherited from <code>Zone</code> .)
<code>GetComputersContainer</code>	Returns the Active Directory object for the Computers node. (Inherited from <code>Zone</code> .)
<code>GetDirectoryEntry</code>	Returns the Active Directory object for the zone. (Inherited from <code>Zone</code> .)
<code>GetDisplayName</code>	Returns the display name of this zone. (Inherited from <code>Zone</code> .)
<code>GetEffectiveCommands</code>	Returns all the command rights that can be assigned to users in the zone, including inherited rights.
<code>GetEffectiveNetworkAccesses</code>	Returns all the network access rights that can be assigned to users in the zone, including inherited rights.
<code>GetEffectivePamAccesses</code>	Returns all the PAM application access rights that



This method	Does this
	can be assigned to users in the zone, including inherited rights.
<code>GetEffectiveRoles</code>	Returns all the user roles that can be assigned to users in the zone, including inherited roles.
<code>GetEffectiveSshs</code>	Returns all the SSH application access rights that can be assigned to users in the zone, including inherited rights.
<code>GetEffectiveUserUnixProfiles</code>	Returns an enumeration of effective users under this zone.
<code>GetEffectivewindowsApplications</code>	Returns all the Windows application access rights that can be assigned to users in the zone, including rights inherited from zones higher in the hierarchy.
<code>GetEffectivewindowsDesktops</code>	Returns all the Windows desktop access rights that can be assigned to users in the zone, including rights inherited from zones higher in the hierarchy.
<code>GetEffectivewindowsUsers</code>	Returns all the Windows users in the zone, including users inherited from zones higher in the hierarchy.
<code>GetLocalGroupsContainer</code>	Returns the DirectoryEntry of the local groups container.  (Inherited from <code>Zone</code> .)
<code>GetLocalUserUnixProfile</code>	Returns the local UNIX group profile for a specified group name in the zone.  (Inherited from <code>Zone</code> .)
<code>GetLocalUserUnixProfileByDN</code>	Returns a local group profile using the distinguished name (DN) of the profile.  (Inherited from <code>Zone</code> .)
<code>GetLocalGroupUnixProfileByGid (Int32)</code>	Returns the local group profile using the Group Identifier (GID). This method is exposed to the .COM interface.  (Inherited from <code>Zone</code> .)
<code>GetLocalGroupUnixProfiles</code>	Returns a list of the local group profiles in the zone.  (Inherited from <code>Zone</code> .)
<code>GetLocalUsersContainer</code>	Returns the directory entry of the local users container.  (Inherited from <code>Zone</code> .)
<code>GetLocalUserUnixProfile</code>	Returns the local user profile using the specified user name.  (Inherited from <code>Zone</code> .)



This method	Does this
<code>GetLocalUserUnixProfileByDN</code>	Returns the local user profile specified by the distinguished name (DN) of the profile.  (Inherited from <code>Zone</code> .)
<code>GetLocalUserUnixProfileByUid (Int32)</code>	Returns the local user profile using the User Identifier (UID). This method is exposed to the .COM interface  (Inherited from <code>Zone</code> .)
<code>GetLocalUserUnixProfiles</code>	Returns a list of the local user profiles in the zone.  (Inherited from <code>Zone</code> .)
<code>GetNetworkAccess</code>	Returns the specified network access right.
<code>GetNetworkAccesses</code>	Returns all the network access rights that can be assigned to users in the zone.
<code>GetGroupsContainer</code>	Returns the Active Directory object for the Groups container.  (Inherited from <code>Zone</code> .)
<code>GetGroupUnixProfile</code>	Returns the UNIX group profile in this zone for the specified Active Directory group.  (Inherited from <code>Zone</code> .)
<code>GetGroupUnixProfileByDN</code>	Returns the UNIX group profile in this zone for the Active Directory group specified by distinguished name.  (Inherited from <code>Zone</code> .)
<code>GetGroupUnixProfileByName</code>	Returns the UNIX group profile in this zone for the Active Directory group specified by group name.  (Inherited from <code>Zone</code> .)
<code>GetGroupUnixProfiles</code>	Returns an enumeration of the UNIX groups in the zone.  (Inherited from <code>Zone</code> .)
<code>GetImportPendingGroup</code>	Returns the group with the specified ID pending import.  (Inherited from <code>Zone</code> .)
<code>GetImportPendingGroups</code>	Returns an enumeration of groups pending import to this zone.  (Inherited from <code>Zone</code> .)
<code>GetImportPendingUser</code>	Returns the user with the specified ID pending import.  (Inherited from <code>Zone</code> .)



<b>This method</b>	<b>Does this</b>
<b>GetImportPendingUsers</b>	Returns an enumeration of users pending import to this zone.  (Inherited from <a href="#">Zone</a> .)
<b>GetNetworkAccess</b>	VBScript interface to access NSS variables.
<b>GetNSSVariables</b>	VBScript interface to obtain all NSS variable names.
<b>GetPamAccess</b>	Returns the PAM application access right with the specified name.
<b>GetPamAccesses</b>	Returns an enumeration of all the PAM application rights in the zone.
<b>GetPrimaryUser</b>	Returns the primary profile for the specified user.
<b>GetRole</b>	Returns the role with the specified name or GUID.
<b>GetRoleAssignment</b>	Returns the role assignment for the specified role and trustee.
<b>GetRoleAssignmentById</b>	Returns the role assignment for the specified GUID.
<b>GetRoleAssignments</b>	Returns an enumeration of all the role assignments in the zone.
<b>GetRoleAssignmentToAllADUsers</b>	Returns the role assignment given to all Active Directory users who have a specified role.
<b>GetRoleAssignmentToAllUnixUsers</b>	Returns the role assignment given to all UNIX users who have a specified role.
<b>GetRoles</b>	Returns an enumeration of all the roles in the zone.
<b>GetSecondaryUsers</b>	Returns an enumeration of the secondary profiles for the specified user.
<b>GetSshRight</b>	Returns the SSH application access right with the specified name.
<b>GetSshRights</b>	Returns an enumeration of all the SSH application rights in the zone.
<b>GetSubTreeRoleAssignments</b>	Returns all role assignments under this zone, including role assignments for computer roles and computers.
<b>GetUserProfiles</b>	Returns an enumeration of all the user profiles for the specified user.
<b>GetUserRoleAssignments</b>	Returns an enumeration of all the user role assignments in the zone.
<b>GetwindowsApplication</b>	Returns the specified Windows application right.
<b>GetwindowsApplications</b>	Returns all the Windows application rights in the zone.
<b>GetwindowsComputers</b>	Returns all the Windows computers in the zone.
<b>GetwindowsDesktop</b>	Returns the specified Windows desktop right.



This method	Does this
<code>GetWindowsDesktops</code>	Returns all the Windows desktop rights in the zone.
<code>GetUsersContainer</code>	Returns the directory entry of the Users container. (Inherited from <code>Zone</code> .)
<code>GetUserUnixProfileByDN</code>	Returns the UNIX user profile in this zone for the user specified by distinguished name. (Inherited from <code>Zone</code> .)
<code>GetUserUnixProfileByName</code>	Returns the UNIX user profile in this zone for the user specified by user name. (Inherited from <code>Zone</code> .)
<code>GetUserUnixProfiles</code>	Returns an enumeration of all the UNIX user profiles in the zone. (Inherited from <code>Zone</code> .)
<code>GroupUnixProfileExists</code>	Indicates whether the group has a profile in this zone. (Inherited from <code>Zone</code> .)
<code>LocalGroupUnixProfileExists</code>	Indicates whether a UNIX profile exists in the zone for the specified local group. (Inherited from <code>Zone</code> .)
<code>LocalUserUnixProfileExists</code>	Indicates whether a UNIX profile exists in the zone for the specified local user. (Inherited from <code>Zone</code> .)
<code>PrecreateComputerZone</code>	Adds a computer zone to a computer object in this zone.
<code>Refresh</code>	Refreshes the data in this object instance from the data stored in Active Directory. (Inherited from <code>Zone</code> .)
<code>SetNSSVariable</code>	VBScript interface to set the values of NSS variables.
<code>UserUnixProfileExists</code>	Indicates whether the specified user has a profile in this zone. (Inherited from <code>Zone</code> .)

## Properties

The `HierarchicalZone` class provides the following properties:



<b>This property</b>	<b>Does this</b>
<b>AdsiInterface</b>	Gets the IADs interface of the zone object in Active Directory. (Inherited from <a href="#">Zone</a> .)
<b>ADSPATH</b>	Gets the LDAP path to the zone object. (Inherited from <a href="#">Zone</a> .)
<b>AgentlessAttribute</b>	Gets or sets the attribute used to store the password hash for an agentless client. (Inherited from <a href="#">Zone</a> .)
<b>AvailableShells</b>	Gets or sets an enumeration of available user login shells. (Inherited from <a href="#">Zone</a> .)
<b>Cims</b>	Gets the Cims object managing this zone. (Inherited from <a href="#">Zone</a> .)
<b>DefaultGroup</b>	Gets or sets the default group for new users. (Inherited from <a href="#">Zone</a> .)
<b>DefaultHomeDirectory</b>	Gets or sets the default login directory for new users. (Inherited from <a href="#">Zone</a> .)
<b>DefaultShell</b>	Gets or sets the default login shell for new users. (Inherited from <a href="#">Zone</a> .)
<b>DefaultValueZone</b>	Gets or sets the zone to use for default zone values. (Inherited from <a href="#">Zone</a> .)
<b>Description</b>	Gets or sets the description of the zone. (Inherited from <a href="#">Zone</a> .)
<b>FullName</b>	Gets or sets the full name of the zone. (Inherited from <a href="#">Zone</a> .)
<b>GroupAutoProvisioningEnabled</b>	Indicates whether auto-provisioning of group profiles is enabled for the zone. (Inherited from <a href="#">Zone</a> .)
<b>GroupDefaultName</b>	Gets or sets the default group name.
<b>ID</b>	Gets the unique identifier for the zone. (Inherited from <a href="#">Zone</a> .)
<b>IsChild</b>	Indicates whether this is a child zone.
<b>IsGroupDefaultNameDefined</b>	Indicates whether the group default name is



This property	Does this
	defined.
<b>IsHierarchical</b>	Indicates whether this is a hierarchical zone. (Inherited from <b>Zone</b> .)
<b>IsNextGidDefined</b>	Gets or sets whether Next GID value is configured for this zone.
<b>IsNextUidDefined</b>	Gets or sets whether Next UID value is configured for this zone.
<b>IsReadable</b>	Indicates whether this zone object in Active Directory is readable with the current user credentials. (Inherited from <b>Zone</b> .)
<b>IsSFU</b>	Indicates whether the zone uses the Microsoft Services for UNIX (SFU) schema extension. (Inherited from <b>Zone</b> .)
<b>IsTruncateName</b>	Indicates whether this is a <b>TruncateName</b> zone. (Inherited from <b>Zone</b> .)
<b>IsUseAutoPrivateGroupDefined</b>	Determines whether the <b>UseAutoPrivateGroup</b> flag is defined.
<b>IsUserDefaultGecosDefined</b>	Determines whether the user default GECOS is defined in this profile.
<b>IsUserDefaultHomeDirectoryDefined</b>	Determines whether the user default home directory is defined in this profile.
<b>IsUserDefaultNameDefined</b>	Determines whether the user default name is defined in this profile.
<b>IsUserDefaultPrimaryGroupDefined</b>	Determines whether the user default primary group is defined in this profile.
<b>IsUserDefaultRoleDefined</b>	Determines whether the user default role is defined in this profile.
<b>IsUserDefaultShellDefined</b>	Determines whether the user default login shell is defined in this profile.
<b>IsWritable</b>	Indicates whether this zone object is writable using the provided credential. (Inherited from <b>Zone</b> .)
<b>Licenses</b>	Gets or sets the license container for the zone. (Inherited from <b>Zone</b> .)
<b>MasterDomainController</b>	Gets or sets the master domain controller for the zone.



This property	Does this
	(Inherited from <a href="#">Zone</a> .)
<b>MustMaintainADGroupMembership</b>	Indicates whether Active Directory group membership must be maintained. (Inherited from <a href="#">Zone</a> .)
<b>Name</b>	Gets or sets the name of the zone. (Inherited from <a href="#">Zone</a> .)
<b>NextAvailableGID</b>	Gets or sets the next GID to be used when adding a group (32-bit for COM programs). (Inherited from <a href="#">Zone</a> .)
<b>NextAvailableUID</b>	Gets or sets the next UID to be used when adding a user (32-bit for COM programs). (Inherited from <a href="#">Zone</a> .)
<b>NextGID</b>	Gets or sets the next GID to be used when adding a group (64-bit for .NET modules). (Inherited from <a href="#">Zone</a> .)
<b>NextUID</b>	Gets or sets the next UID to be used when adding a user (64-bit for .NET modules). (Inherited from <a href="#">Zone</a> .)
<b>NISDomain</b>	Gets or sets the NIS domain associated with this SFU zone. (Inherited from <a href="#">Zone</a> .)
<b>NssVariables</b>	Gets the map of profile variables.
<b>Parent</b>	Gets or sets the parent of this zone.
<b>ReservedGID</b>	Gets or sets the list of GIDs not to be used when adding groups. (Inherited from <a href="#">Zone</a> .)
<b>ReservedUID</b>	Gets or sets the list of UIDs not to be used when adding users. (Inherited from <a href="#">Zone</a> .)
<b>Schema</b>	Gets the schema of the zone. (Inherited from <a href="#">Zone</a> .)
<b>SFUDomain</b>	Gets or sets the Active Directory domain associated with this SFU zone for retrieving SFU information. (Inherited from <a href="#">Zone</a> .)
<b>UseAppleGid</b>	Determines whether to use the Apple algorithm to automatically generate the GID when adding a



This property	Does this
	group. The Apple algorithm is based on the globally unique identifier (GUID) for the object.
<code>UseAppleUid</code>	Determines whether to use the Apple algorithm to automatically generate the UID when adding a user. The Apple algorithm is based on the globally unique identifier (GUID) for the object.
<code>UseAutoGid</code>	Determines whether to use the Centrify algorithm to automatically generate the GUID when adding a group. The Centrify algorithm is based on the security identifier (SID) for the object.
<code>UseAutoPrivateGroup</code>	Determines whether this zone defaults to use an auto private group when adding a zone user.
<code>UseAutoUid</code>	Determines whether to use the Centrify algorithm to automatically generate the UID when adding a user. The Centrify algorithm is based on the security identifier (SID) for the object.
<code>UseNextGid</code>	Determines whether to use the <code>NextGUID</code> property when adding a group.
<code>UseNextUid</code>	Determines whether to use the <code>NextUID</code> property when adding a user.
<code>UserAutoProvisioningEnabled</code>	Indicates whether auto-provisioning of user profiles is enabled for the zone.  (Inherited from <code>Zone</code> .)
<code>UserDefaultGecos</code>	Gets or sets the default GECOS field for new user profiles.
<code>UserDefaultGid</code>	Gets or sets the user default GUID when adding a new user profile.
<code>UserDefaultName</code>	Gets or sets the default user name for a new user profile.
<code>UserDefaultPrimaryGroup</code>	Gets or sets the user default GUID for new user profiles; for use in VBScript scripts.
<code>UserDefaultRole</code>	Gets or sets the default role for a new user profile.
<code>version</code>	Gets the version number of the data schema.  (Inherited from <code>Zone</code> .)

## AddAccessGroup

Adds an empty role assignment to a group.



## Syntax

`IHzRoleAssignment AddAccessGroup(DirectoryEntry groupDE)`

`IHzRoleAssignment AddAccessGroup(SearchResult groupSR)`

`IHzRoleAssignment AddAccessGroup(string groupDn)`

`IHzRoleAssignment AddAccessGroup(IAdsGroup groupIAds)`

## Parameters

Specify one of the following parameters when using this method.

Parameter	Description
<code>groupDE</code>	The directory entry for the group.
<code>groupSr</code>	The directory entry for the group specified as a search result.
<code>groupDn</code>	The group specified as a distinguished name.
<code>groupIads</code>	The IADs interface to the group.

## Return value

The computer role assignment.

## Discussion

The role assignment is not stored in Active Directory until you call the `Commit` method.

The `AddAccessGroup(DirectoryEntry groupDE)` and `AddAccessGroup(SearchResult groupSR)` methods are available only for .NET-based programs; call `User` for VBScript.

## Exceptions

`AddAccessGroup` may throw one of the following exceptions:

- `ApplicationException` if the specified parameter is not a group or the method cannot find the group.
- `ArgumentNullException` if you pass a `null` parameter.

## Example

The following code sample illustrates using the `AddAccessGroup` and `GetAccessGroup` methods in a script:

• • • • •

```
...
// Get the zone object
IHierarchicalZone objZone =
cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
    return;
}
IRole role = objZone.GetRole(strRole);
if (role == null)
{
    Console.WriteLine(strRole + " does not exist in zone.");
}
else if (objZone.GetAccessGroup(role, strGroup) != null)
{
    Console.WriteLine("Role assignment already exist.");
}
else
{
    // assign a role to the group
    IRoleAssignment zag = objZone.AddAccessGroup(strGroup);
    zag.Role = role;
    zag.Commit();
}
...
```

## AddComputerRole

Creates a computer role under this zone.

### Syntax

```
IComputerRole AddComputerRole(string name)
```

### Parameters

Specify the following parameter when using this method:

Parameter	Description
name	The name of the computer role you want to add.

### Return value

The computer role you added.



## AddGroupPartialProfile

Adds a partial profile for the specified group to the zone.

### Syntax

```
IHierarchicalGroup AddGroupPartialProfile(DirectoryEntry groupDE)
```

```
IHierarchicalGroup AddGroupPartialProfile(SearchResult groupSR)
```

```
IHierarchicalGroup AddGroupPartialProfile(string groupDn)
```

```
IHierarchicalGroup AddGroupPartialProfile(IAdsGroup groupIAds)
```

### Parameters

Specify one of the following parameters when using this method.

Parameter	Description
groupDE	The directory entry for the group for which you want a partial profile.
groupSR	The directory entry for a group specified as a search result.
groupDn	The group specified as a distinguished name.
groupIads	The IADs interface to the group.

### Return value

The hierarchical group object that represents the group profile.

### Discussion

This method creates a new group profile with values set for the `Cims` and `Group` properties. If the zone is an SFU zone, then this method also sets a value for the `NISDomain` property. You can then add other properties to the profile.

The profile is not stored in Active Directory until you call the `Commit` method.

The `AddAccessGroup(DirectoryEntry groupDE)` and `AddAccessGroup(SearchResult groupSR)` methods are available only for .NET-based programs; call `User` for VBScript.

### Exceptions

If you pass a null parameter, `AddGroupPartialProfile` throws the exception `ArgumentNullException`.



## Example

The following code sample illustrates using the `AddGroupPartialProfile` method in a script:

```
...
// Get the zone object
IHierarchicalZone objZone = cims.GetZoneByPath("cn=" + strZone +
", " + strContainerDN) as
    IHierarchicalZone;
// Load the unix profiles associated with the group
// Determine if the specified group is already a member of the
zone.
// This method will either return a blank objGroupUnixProfile
// or one containing data
IGroupUnixProfile objGroupUnixProfile;
if (objZone.GetGroupUnixProfileByName(strUnixGroup) == null)
{
    // Add this zone to the group
    objGroupUnixProfile = objZone.AddGroupPartialProfile
(strGroup);
    objGroupUnixProfile.Name = strUnixGroup;
// Save
objGroupUnixProfile.Commit();
}
...
```

## AddRoleAssignment

Adds an empty role assignment to the zone.

### Syntax

```
IRoleAssignment AddRoleAssignment()
```

### Return value

An empty role assignment object. This role assignment is not stored in Active Directory until you call the `RoleAssignment.Commit` method.

## AddLocalGroupPartialProfile

Adds a partial profile for the specified group to the zone.

### Syntax

```
IHierarchicalUser AddLocalGroupPartialProfile(string groupName)
```

• • • • •

## Parameters

Specify groupName; the name of the local group.

## Return value

The hierarchical group object that represents the local group profile.

## Exceptions

If you pass a null parameter, AddLocalGroupPartialProfile throws the exception ArgumentException.

## AddLocalUserPartialProfile

Adds a partial profile for the specified user to the zone.

## Syntax

```
IHierarchicalUser AddLocalUserPartialProfile(string userName)
```

## Parameters

Specify userName; the user name of the local user.

## Return value

The hierarchical user object that represents the local user profile.

## Exceptions

If you pass a null parameter, AddLocalUserPartialProfile throws the exception ArgumentException.

## AddUserPartialProfile

Adds a partial profile for the specified user to the zone.

## Syntax

```
IHierarchicalUser AddUserPartialProfile(DirectoryEntry userDE)
```

```
IHierarchicalUser AddUserPartialProfile(SearchResult usersR)
```



```
IHierarchicalUser AddUserPartialProfile(string userDn)  
IHierarchicalUser AddUserPartialProfile(IAdsUser userIAds)
```

## Parameters

Specify one of the following parameters when using this method.

Parameter	Description
userDE	The directory entry for the user for which you want a partial profile.
userSr	The directory entry for a user specified as a search result.
userDn	The user specified as a distinguished name.
userIads	The IADs interface to the user.

## Return value

The hierarchical user object that represents the user profile.

## Discussion

This method creates a new user profile with values set for the Cims and User properties. If the zone is an SFU zone, then this method also sets a value for the NISDomain property. You can then add other properties to the profile.

The profile is not stored in Active Directory until you call the **Commit** method.

## Exceptions

If you pass a null parameter, `AddUserPartialProfile` throws the exception `ArgumentNullException`.

## Example

The following code sample illustrates using the `AddUserPartialProfile` method in a script:

```
...  
// Create a CIMS object to interact with AD  
ICims cims = new Cims();  
// Note the lack of the cims.connect function.  
// By default, this application will use the connection to the  
domain controller  
// and existing credentials from the computer already logged in.  
// Get the zone object  
IHierarchicalZone objZone =  
cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as  
IHierarchicalZone;  
if (objZone == null)
```

• • • • •

```
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
    return;
}
IUser objUser = cims.GetUserByPath(strUser);
if (objUser == null)
{
    Console.WriteLine("User " + strUser + " does not exist.");
    return;
}
IHierarchicalUser objUserUnixProfile = (IHierarchicalUser)
    objZone.GetUserUnixProfile(objUser);
if (objUserUnixProfile == null)
{
    //New user for the zone
    objUserUnixProfile = objZone.AddUserPartialProfile(strUser);
}
IRole objRole = objZone.GetRole(strRole);
if (objRole == null)
{
    Console.WriteLine("Role " + strRole + " does not exist.");
    return;
}
IRoleAssignment asg = objUserUnixProfile.GetUserRoleAssignment
(objRole);
if (asg != null)
{
    Console.WriteLine("Assignment already exist.");
    return;
}
else
{
    // assigning role to user
    asg = objUserUnixProfile.AddUserRoleAssignment();
    asg.Role = objZone.GetRole(strRole);
    asg.Commit();
}
...

```

## CreateCommand

Creates a command right for the zone.

### Syntax

```
ICommand CreateCommand ()
```

### Return value

A command right for the zone.



## Discussion

A command right controls who has permission to run a specific command in a zone.

The profile is not stored in Active Directory until you call the `Commit` method.

## Example

The following code sample illustrates using the `CreateCommand` method in a script:

```
...
// Get the zone object
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
}
else
{
    ICommand objCmd = objZone.GetCommand(strCmd);
    if (objCmd != null)
    {
        Console.WriteLine("Command " + strCmd + " already
exists.");
    }
    else
    {
        objCmd = objZone.CreateCommand();
        objCmd.Name = strCmd;
        objCmd.CommandPattern = strPattern;
        objCmd.Description = "optional description";
        objCmd.Commit();
        Console.WriteLine("Command " + strCmd + " was created
successfully.");
    }
}
...
```

## CreateNetworkAccess

Creates a network application access right.

## Syntax

```
INetworkAccess CreateNetworkAccess ()
```



## Return value

A network application access right for the zone.

## Discussion

A network access right enables a user to run an application on a remote computer as another user. For example, a network access right can give a user the ability to run as an SQL Administrator on a remote server.

The right is not stored in Active Directory until you call the `Commit` method.

## Example

The following code sample illustrates using the `CreateNetworkAccess` method in a script:

```
...
// Get the zone object
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
}
else
{
    INetworkAccess objNetworkAccess = objZone.GetNetworkAccess
(strName);
    if (objNetworkAccess != null)
    {
        Console.WriteLine("NetworkAccess " + strName + " already
exist.");
    }
    else
    {
        objNetworkAccess = objZone.CreateNetworkAccess();
        objNetworkAccess.Name = strName;
        objNetworkAccess.RunAsType = windowsRunAsType.User;
        objNetworkAccess.Priority = 0;
        objNetworkAccess.Description = "optional description";
        string userPath = DirectoryServices.GetLdapPathFromDN
(cims.Server, strUser);
        DirectoryEntry userEntry =
DirectoryServices.GetDirectoryEntry(userPath,
    cims.UserName, cims.Password);
        SecurityIdentifier m_userSid = new
SecurityIdentifier(DirectoryServices.GetStringsSid
(userEntry));
        objNetworkAccess.RunAsList = new List<SecurityIdentifier>
{ m_userSid };
        objNetworkAccess.Commit();
        Console.WriteLine("NetworkAccess " + strName + " is
```

• • • • •

```
created successfully.");  
    }  
}  
...
```

## CreatePamAccess

Creates a PAM application access right.

### Syntax

```
IPam CreatePamAccess ()
```

### Return value

A PAM application access right for the zone.

### Discussion

A PAM (Pluggable Authentication Module) application right gives a user the ability to access the authorized PAM-enabled application.

The right is not stored in Active Directory until you call the `Commit` method.

### Example

The following code sample illustrates using the `CreatePamAccess` method in a script:

```
...  
// Get the zone object  
IHierarchicalZone objZone =  
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as  
IHierarchicalZone;  
if (objZone == null)  
{  
    Console.WriteLine("Zone " + strZone + " does not exist.");  
}  
else  
{  
    IPam objPam = objZone.GetPamAccess(strName);  
    if (objPam != null)  
    {  
        Console.WriteLine("PAM " + strName + " already exists.");  
    }  
    else  
    {  
        objPam = objZone.CreatePamAccess();  
        objPam.Name = strName;  
        objPam.Application = strApp;  
    }  
}
```

• • • • •

```
        objPam.Description = "optional description";
        objPam.Commit();
    }
}
...
```

## CreateRole

Creates a role in the zone.

### Syntax

```
IRole CreateRole (string name)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the role.

### Return value

A role with the specified name.

### Discussion

The role is not stored in Active Directory until you call the `Commit` method.

### Example

The following code sample illustrates using the `CreateRole` method in a script:

```
...
// Get the zone object
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
// create the role
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
}
else
{
    IRole role = objZone.CreateRole(strRole);
    role.Description = "optional description";
    role.Commit();
}
```

• • • • •

```
}  
...
```

## CreateSshRight

Creates an SSH application access right.

### Syntax

```
Issh CreateSshRight ()
```

### Return value

An SSH application access right for the zone.

### Discussion

An SSH (Secure Shell) application right gives a user the ability to access the authorized SSH-enabled application.

The right is not stored in Active Directory until you call the `Commit` method.

## CreateWindowsApplication

Creates a Windows application access right.

### Syntax

```
IwindowsApplication CreateWindowsApplication ()
```

### Return value

A Windows application access right for the zone.

### Discussion

A Windows application right gives a user the ability to access the authorized Windows application.

The right is not stored in Active Directory until you call the `Commit` method.

• • • • •

## Example

The following code sample illustrates using the `CreateWindowsApplication` method in a script:

```
...
// Get the zone object
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
}
else
{
    IWindowsApplication objWindowsApplication =
objZone.GetWindowsApplication(strName);
    if (objWindowsApplication != null)
    {
        Console.WriteLine("WindowsApplication " + strName + "
already exists.");
    }
    else
    {
        objWindowsApplication = objZone.CreateWindowsApplication
();
        objWindowsApplication.Name = strName;
        objWindowsApplication.RunAsType = WindowsRunAsType.Self;
        objWindowsApplication.Priority = 0;
        objWindowsApplication.Description = "optional
description";
        objWindowsApplication.Command = strApplication;
        objWindowsApplication.Commit();
        Console.WriteLine("Windows Application " + strName + "
has been created
        successfully.");
    }
}
...
```

## CreateWindowsDesktop

Creates a Windows Desktop access right.

### Syntax

```
IWindowsDesktop CreateWindowsDesktop ()
```

### Return value

A Windows desktop access right for the zone.



## Discussion

A Windows desktop right provides a complete desktop that behaves as if the user had logged in as specific privileged user. For example, if you have an SQL Administrator login, you can give an ordinary user an SQL Administrator desktop so they can operate in that role when necessary.

The right is not stored in Active Directory until you call the `Commit` method.

## Example

The following code sample illustrates using the `CreateWindowsDesktop` method in a script:

```
...
// Get the zone object
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
    IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
}
else
{
    IWindowsDesktop objWindowsDesktop = objZone.GetWindowsDesktop
    (strName);
    if (objWindowsDesktop != null)
    {
        Console.WriteLine("Windows Desktop " + strName + " already
exists.");
    }
    else
    {
        objWindowsDesktop = objZone.CreateWindowsDesktop();
        objWindowsDesktop.Name = strName;
        objWindowsDesktop.RunAsType = WindowsRunAsType.Self;
        objWindowsDesktop.Priority = 0;
        objWindowsDesktop.Description = "optional description";
        objWindowsDesktop.Commit();
        Console.WriteLine("Windows Desktop " + strName + " has
been created
        successfully.");
    }
}
...
```

## GeneratePredefinedRights

Generates predefined SSH and PAM rights in this zone.



## Syntax

```
Void GeneratePredefinedRights ()
```

## Discussion

This method calls the `CreateSshRight` and `CreatePamAccess` methods for a predefined list of SSH and PAM applications. You can call the `GetSshRights` and `GetPamAccesses` methods to get lists of the SSH and PAM rights that have been created in the zone. The rights are stored in Active Directory; you do not have to call the `Commit` method.

## GeneratePredefinedRoles

Generates predefined roles.

## Syntax

```
void GeneratePredefinedRoles ()
```

## Discussion

This method calls the `CreateRole` method for a predefined list of user roles, such as the Windows Login and UNIX Login roles. You can call the `GetRoles` method to get a list of the roles that have been created in the zone. The roles are stored in Active Directory; you do not have to call the `Commit` method.

## GetAccessGroup

Gets a user group assigned to this zone given a specific role.

## Syntax

```
IHzRoleAssignment GetAccessGroup(IRole role, DirectoryEntry  
group)
```

```
IHzRoleAssignment GetAccessGroup(IRole role, SearchResult  
groupSr)
```

```
IHzRoleAssignment GetAccessGroup(IRole role, string groupDn)
```

```
IHzRoleAssignment GetAccessGroup(IRole role, IADsGroup groupIAds)
```



## Parameters

Specify the following parameter when using this method:

Parameter	Description
<code>role</code>	The role of the group.

Specify one of the following parameters when using this method.

Parameter	Description
<code>group</code>	The directory entry for the group.
<code>groupSr</code>	The directory entry for the group specified as a search result.
<code>groupDn</code>	The group specified as a distinguished name.
<code>groupIads</code>	The IADs interface to the group.

## Return value

The computer role assignment that includes the specified group (`IHzRoleAssignment.TrusteeType==Group`).

## Discussion

Any number of user groups can be assigned to a computer role and each of those groups can have more than one role. Use this method to get the computer role assignment for a specific group and role.

The `GetAccessGroup(IRole role, DirectoryEntry group)` and `GetAccessGroup(IRole role, SearchResult groupSr)` methods are available only for .NET-based programs; call `GetRoleAssignment` for VBScript.

## Exceptions

`GetAccessGroup` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is null.
- `ApplicationException` if the parameter value is not a valid user; or if it failed to create a role assignment because it cannot find the user.

## Example

See `AddAccessGroup` for an example of using the `GetAccessGroup` method in a script:



## GetAccessGroups

Returns the computer roles assigned to this zone.

### Syntax

```
IRoleAssignments GetAccessGroups()
```

### Return value

The collection of computer roles. Enumerate this object to get all of the `IAZRoleAssignment` objects in this zone.

## GetChildZones

Returns the child zones of this zone.

### Syntax

```
IEnumerable GetChildZones()
```

### Return value

The collection of child zones.

### Exceptions

`GetChildZones` throws an `ApplicationException` if it can't find the zone. For example, `GetChildZones` throws an `ApplicationException` if the zone has been removed or the server is not available.

## GetCommand

Returns the command right with a specified name or GUID.

### Syntax

```
ICommand GetCommand (string name)
```

```
ICommand GetCommand (Guid id)
```



## Parameter

Specify one of the following parameters when using this method:

Parameter	Description
name	The name of the command.
id	The GUID of the command.

## Return value

A command right with the specified name or GUID, or null if no match is found.

## Exceptions

GetCommand may throw one of the following exceptions:

- `ApplicationException` if it can't find authorization data for the zone or if it failed to get the command right (see the message returned by the exception for the reason).
- `ArgumentException` if the name or id parameter is null or empty.

## Example

The following code sample illustrates using the GetCommand method in a script:

```
...
// Get the zone object
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
    IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
}
else
{
    ICommand objCmd = objZone.GetCommand(strCmd);
    if (objCmd != null)
    {
        Console.WriteLine("Command " + strCmd + " already
exists.");
    }
    else
    {
        objCmd = objZone.CreateCommand();
        objCmd.Name = strCmd;
        objCmd.CommandPattern = strPattern;
        objCmd.Description = "optional description";
        objCmd.Commit();
    }
}
```

• • • • •

```
}  
...
```

## GetCommands

Returns all the command rights in the zone.

### Syntax

```
ICommands GetCommands()
```

### Return value

The collection of commands in the zone.

## GetComputerRole

Returns the computer role with a specified name.

### Syntax

```
IComputerRole GetComputerRole (string name)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the computer role.

### Return value

The computer role with the specified name, or null if no match is found.

### Exceptions

GetComputerRole throws an `ApplicationException` if it can't find authorization data for the zone or if it failed to get the computer role (see the message returned by the exception for the reason).

• • • • •

## Example

The following code sample illustrates using the `GetComputerRole` method in a script:

```
...
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
}
else
{
    IComputerRole compRole = objZone.GetComputerRole(strName);
    if (compRole != null)
    {
        Console.WriteLine("Computer role " + strName + " already
exist.");
    }
    else
    {
        compRole = objZone.AddComputerRole(strName);
        compRole.Group = strGroup;
        compRole.Validate();
        compRole.Commit();
        Console.WriteLine("Computer role " + strName + " is
created successfully.");
    }
}
...
```

## GetComputerRoles

Returns all the computer roles in the zone.

### Syntax

```
IComputerRoles GetComputerRoles()
```

### Return value

The collection of computer roles in the zone.



## GetEffectiveCommands

Returns all the command rights that can be assigned to users in the zone, including rights inherited from zones higher in the hierarchy.

### Syntax

```
ICommands GetEffectiveCommands()
```

### Return value

The collection of effective command rights in the zone.

### Exceptions

`GetEffectiveCommands` throws an `ApplicationException` if it failed to get the effective command rights (see the message returned by the exception for the reason).

## GetEffectiveNetworkAccesses

Returns all the network access rights that can be assigned to users in the zone, including rights inherited from zones higher in the hierarchy.

### Syntax

```
INetworkAccesses GetEffectiveNetworkAccesses()
```

### Return value

The collection of effective network access rights in the zone.

### Exceptions

`GetEffectiveNetworkAccesses` throw an `ApplicationException` if it failed to get the effective network access rights (see the message returned by the exception for the reason).



## GetEffectivePamAccesses

Returns all the PAM application access rights that can be assigned to users in the zone, including rights inherited from zones higher in the hierarchy.

### Syntax

```
IPams GetEffectivePamAccesses()
```

### Return value

The collection of effective PAM application access rights in the zone.

### Exceptions

`GetEffectivePamAccesses` throws an `ApplicationException` if it failed to get the effective command rights (see the message returned by the exception for the reason).

## GetEffectiveRoles

Returns all the roles that can be assigned to users in the zone, including roles inherited from zones higher in the hierarchy.

### Syntax

```
IRoles GetEffectiveRoles()
```

### Return value

The collection of effective roles in the zone.

### Exceptions

`GetEffectiveRoles` throws an `ApplicationException` if it failed to get the effective command rights (see the message returned by the exception for the reason).



## GetEffectiveSshs

Returns all the SSH application access rights that can be assigned to users in the zone, including rights inherited from zones higher in the hierarchy.

### Syntax

```
ISshs GetEffectiveSshs()
```

### Return value

The collection of effective SSH application access rights in the zone.

### Exceptions

GetEffectiveSshs throws an `ApplicationException` if it fails to get the effective command rights (see the message returned by the exception for the reason).

## GetEffectiveUserUnixProfiles

Returns all the users in the zone, including users inherited from zones higher in the hierarchy.

### Syntax

```
IUserUnixProfiles GetEffectiveUserUnixProfiles()
```

### Return value

The collection of effective users in the zone.

## GetEffectiveWindowsApplications

Returns all the Windows application access rights that can be assigned to users in the zone, including rights inherited from zones higher in the hierarchy.

### Syntax

```
IWindowsApplications GetEffectiveWindowsApplications()
```



## Return value

The collection of effective Windows application access rights in the zone.

## Exceptions

`GetEffectivewindowsApplications` throws an `ApplicationException` if it fails to get the effective access rights (see the message returned by the exception for the reason).

## GetEffectiveWindowsDesktops

Returns all the Windows desktop access rights that can be assigned to users in the zone, including rights inherited from zones higher in the hierarchy.

## Syntax

```
IwindowsDesktops GetEffectivewindowsDesktops()
```

## Return value

The collection of effective Windows desktop access rights in the zone.

## Exceptions

`GetEffectivewindowsDesktops` throws an `ApplicationException` if it failed to get the effective access rights (see the message returned by the exception for the reason).

## GetEffectiveWindowsUsers

Returns all the Windows users in the zone, including users inherited from zones higher in the hierarchy.

## Syntax

```
IwindowsUsers GetEffectivewindowsUsers()
```

## Return value

The collection of effective Windows users in the zone.



## GetNetworkAccess

Returns the specified network access right.

### Syntax

```
INetworkAccess GetNetworkAccess (string name)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the access right.

### Return value

The network access right with the specified name.

### Exceptions

GetNetworkAccess may throw one of the following exceptions:

- `ArgumentException` if the parameter value is `null` or empty.
- `ApplicationException` if cannot find authorization for the zone, or if it failed to get the network access right (see the message returned by the exception for the reason).

### Example

For an example of the use of the `GetNetworkAccess` method, see [CreateNetworkAccess](#).

## GetNetworkAccesses

Returns all the network access rights that can be assigned to users in the zone.

### Syntax

```
INetworkAccesses GetNetworkAccesses()
```

• • • • •

## Return value

The collection of NSS variable names.

## Discussion

This method returns only the network access rights assigned in the current zone. Call `GetEffectiveNetworkAccesses` to return all the network access rights including those inherited from zones higher in the hierarchy.

## GetNSSVariable

Returns the specified NSS environment variable; VBScript only.

## Syntax

```
string GetNssVariable (string name)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the variable.

## Return value

The value of the variable, or null if name is not a defined variable.

## GetNSSVariables

Returns the names of all NSS variables; VBScript only.

## Syntax

```
IEnumerable GetNSSVariables()
```

## Return value

The collection of NSS variable names.



## GetPamAccess

Returns the specified PAM application access right.

### Syntax

```
IPam GetPamAccess (string name)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the PAM access right.

### Return value

The PAM application access right with the specified name, or `null` if name is not in use.

### Exceptions

`GetPamAccess` may throw one of the following exceptions:

- `ApplicationException` if it can't find authorization data for the zone or if it failed to get the PAM application access right (see the message returned by the exception for the reason).
- `ArgumentException` if the name parameter is `null` or empty.

### Example

For sample code using the `GetPamAccess` method in a script, see [CreatePamAccess](#).

## GetPamAccesses

Returns all the PAM application access rights in the zone.

### Syntax

```
IPams GetPamAccesses()
```



## Return value

The collection of PAM application access rights in the zone.

## GetPrimaryUser

Returns the primary profile for the specified user.

## Syntax

```
IHierarchicalUser GetPrimaryUser(DirectoryEntry userDE)
```

```
IHierarchicalUser GetPrimaryUser(SearchResult userSR)
```

```
IHierarchicalUser GetPrimaryUser(string userDn)
```

```
IHierarchicalUser GetPrimaryUser(IAdsUser userIAds)
```

## Parameters

Specify one of the following parameters when using this method.

Parameter	Description
userDE	The directory entry for the user for which you want the primary profile.
userSR	The directory entry for a user specified as a search result.
userDn	The user specified as a distinguished name.
userIads	The IADs interface to the user.

## Return value

The hierarchical user object that represents the user profile.

## Discussion

The primary profile is the profile at the highest level in the zone hierarchy where the user's profile is defined. All or part of the primary profile can be overridden by secondary profiles farther down in the hierarchy.

The `GetPrimaryUser(DirectoryEntry userDE)` and `GetPrimaryUser(SearchResult userSR)` methods are available only for .NET-based programs.



## Exceptions

`GetPrimaryUser` throws an `ArgumentNullException` if the specified parameter value is `null` or empty, or if the user does not exist.

## GetRole

Returns the role with a specified name or GUID.

## Syntax

```
IRole GetRole (string name)
```

```
IRole GetRole (Guid id)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the role.
id	The GUID of the role.

## Return value

The role with the specified name, or `null` if no match is found.

## Exceptions

`GetRole` may throw one of the following exceptions:

- `ApplicationException` if it can't find authorization data for the zone or if it failed to get the role (see the message returned by the exception for the reason).
- `ArgumentException` if the parameter is `null` or empty.

## Example

The following code sample illustrates using the `GetRole` method in a script:

```
...  
// Get the zone object  
IHierarchicalZone objZone =  
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as  
IHierarchicalZone;
```

• • • • •

```
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
    return;
}
IRole role = objZone.GetRole(strRole);
if (role == null)
{
    Console.WriteLine(strRole + " does not exist in zone.");
}
else if (objZone.GetAccessGroup(role, strGroup) != null)
{
    Console.WriteLine("Role assignment already exist.");
}
else
{
    // assign a role to the group
    IRoleAssignment zag = objZone.AddAccessGroup(strGroup);
    zag.Role = role;
    zag.Commit();
}
...
```

## GetRoleAssignment

Returns a role assignment given a role and trustee.

### Syntax

```
IRoleAssignment GetRoleAssignment (IRole role, string dn)
```

### Parameter

Specify the following parameters when using this method.

Parameter	Description
role	The role for which you want the assignment.
dn	The distinguished name of the user or group to whom the role is assigned.

### Return value

The role assignment, or null if no match is found.

### Exceptions

GetRoleAssignment throws an ArgumentException if either parameter is null or empty.



## GetRoleAssignmentById

Returns a role assignment given an ID.

### Syntax

```
IRoleAssignment GetRoleAssignmentById (Guid id)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
id	The GUID of the role assignment.

### Return value

The role assignment, or null if no match is found.

### Exceptions

GetRoleAssignment throws an `ArgumentException` if the parameter is empty.

## GetRoleAssignments

Returns all the role assignments in the zone.

### Syntax

```
IRoleAssignments GetRoleAssignments()
```

### Return value

The collection of role assignments in the zone.

## GetRoleAssignmentToAllADUsers

Returns the role assignment given to all Active Directory users who have a specified role.



## Syntax

```
IRoleAssignment GetRoleAssignmentToAllADUsers(IRole role)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>role</code>	The user role for which you want the role assignments.

## Return value

The role assignment for the specified role.

## Exceptions

`GetRoleAssignmentToAllADUsers` throws an `ArgumentNullException` if the parameter is `null`.

## GetRoleAssignmentToAllUnixUsers

Returns the role assignment given to all UNIX users who have a specified role.

## Syntax

```
IRoleAssignment GetRoleAssignmentToAllUnixUsers(IRole role)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>role</code>	The user role for which you want the role assignments.

## Return value

The role assignment for the specified role.



## Discussion

This method returns the role assignment for local UNIX users with the specified role,.

## Exceptions

`GetRoleAssignmentToAllUnixUsers` throws an `ArgumentNullException` if the parameter is `null`.

## GetRoles

Returns all the roles in the zone.

## Syntax

```
IRoles GetRoles()
```

## Return value

The collection of computer roles in the zone.

## GetSecondaryUsers

Returns the secondary profiles for the specified user.

## Syntax

```
IUserUnixProfiles GetSecondaryUsers(DirectoryEntry userDE)
```

```
IUserUnixProfiles GetSecondaryUsers(SearchResult userSR)
```

```
IUserUnixProfiles GetSecondaryUsers(string userDn)
```

```
IUserUnixProfiles GetSecondaryUsers(IAdsUser userIAds)
```

## Parameters

Specify one of the following parameters when using this method.

---

Parameter	Description
<code>userDE</code>	The directory entry for the user for which you want the secondary profiles.
<code>userSR</code>	The directory entry for a user specified as a search result.

---



---

Parameter	Description
userDn	The user specified as a distinguished name.
userIads	The IADs interface to the user.

---

## Return value

The collection of secondary user UNIX profiles.

## Discussion

The primary profile is the profile at the highest level in the zone hierarchy where the user's profile is defined. All or part of the primary profile can be overridden by secondary profiles farther down in the hierarchy.

The `GetSecondaryUser(DirectoryEntry userDE)` and `GetSecondaryUser(SearchResult userSr)` methods are available only for .NET-based programs.

## Exceptions

`GetSecondaryUsers` throws an `ArgumentNullException` if the parameter is `null` or the user does not exist.

## GetSshRight

Returns the specified SSH application access right.

## Syntax

```
ISsh GetSshRight (string name)
```

## Parameter

Specify the following parameter when using this method:

---

Parameter	Description
name	The name of the SSH access right.

---



## Return value

The SSH application access right with the specified name, or `null` if name is not in use.

## Exceptions

`getSshRight` may throw one of the following exceptions:

- `ApplicationException` if it can't find authorization data for the zone or if it failed to get the right (see the message returned by the exception for the reason).
- `ArgumentException` if the parameter is `null` or empty.

## GetSshRights

Returns all the SSH application access rights in the zone.

### Syntax

```
ISshs GetSshRights()
```

## Return value

The collection of SSH application access rights in the zone.

## GetSubTreeRoleAssignments

Returns all the role assignments under this zone, including role assignments for computer roles and computers.

### Syntax

```
IRoleAssignments GetSubTreeRoleAssignments()
```

## Return value

The collection of role assignments in the zone.



## Exceptions

`GetSubTreeRoleAssignments` throws an `ApplicationException` if the method fails to find the authorization store.

## GetUserProfiles

Returns all the profiles for the specified user.

### Syntax

```
IUserUnixProfiles GetUserProfiles(DirectoryEntry userDE)
```

```
IUserUnixProfiles GetUserProfiles(SearchResult userSR)
```

```
IUserUnixProfiles GetUserProfiles(string userDn)
```

```
IUserUnixProfiles GetUserProfiles(IAdsUser userIAds)
```

### Parameters

Specify one of the following parameters when using this method.

Parameter	Description
<code>userDE</code>	The directory entry for the user for which you want the profiles.
<code>userSr</code>	The directory entry for a user specified as a search result.
<code>userDn</code>	The user specified as a distinguished name.
<code>userIads</code>	The IADs interface to the user.

### Return value

The collection of user UNIX profiles.

### Exceptions

`GetUserProfiles` throws an `ArgumentNullException` if the parameter is `null` or the user does not exist.

## GetUserRoleAssignments

Returns all the user role assignments in the zone, or for a specific user in the zone.



## Syntax

```
IRoleAssignments GetUserRoleAssignments()  
IRoleAssignments GetUserRoleAssignments(DirectoryEntry userDE)  
IRoleAssignments GetUserRoleAssignments(SearchResult userSR)  
IRoleAssignments GetUserRoleAssignments(string userDn)  
IRoleAssignments GetUserRoleAssignments(IAdsUser userIAds)  
IRoleAssignments GetUserRoleAssignments(IUser user)
```

## Parameters

Specify no parameters to return all the role assignments in the zone.

Specify one of the following parameters to return all the role assignments for a specific user:

Parameter	Description
userDE	The directory entry for the user for which you want the role assignments.
userSr	The directory entry for a user specified as a search result.
userDn	The user specified as a distinguished name.
userIads	The IADs interface to the user.
user	The user specified as a CIMS user object.

## Return value

The collection of role assignments as `IRoleAssignment` objects.

## Exceptions

`GetUserRoleAssignments` throws an `ArgumentNullException` if the required parameter is `null` or empty.

## GetWindowsApplication

Returns the specified Windows application right.

## Syntax

```
IWindowsApplication GetWindowsApplication (string name)
```



## Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the Windows application.

## Return value

The Windows application right with the specified name, or `null` if name is not in use.

## Exceptions

`GetWindowsApplication` may throw one of the following exceptions:

- `ApplicationException` if it can't find authorization data for the zone or if it failed to get the right (see the message returned by the exception for the reason).
- `ArgumentException` if the parameter is `null` or empty.

## GetWindowsApplications

Returns all the Windows application rights in the zone.

### Syntax

```
IWindowsApplications GetWindowsApplications()
```

### Return value

The collection of Windows application rights in the zone.

## GetWindowsComputers

Returns all the Windows computers in the zone.

### Syntax

```
IComputers GetWindowsComputers()
```



## Return value

The collection of Windows computers in the zone.

## GetWindowsDesktop

Returns the specified Windows desktop right.

## Syntax

```
IWindowsDesktop GetWindowsDesktop (string name)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the Windows desktop.

## Return value

The Windows desktop right with the specified name, or `null` if name is not in use.

## Exceptions

`GetWindowsDesktop` may throw one of the following exceptions:

- `ApplicationException` if it can't find authorization data for the zone or if it failed to get the right (see the message returned by the exception for the reason).
- `ArgumentException` if the parameter is `null` or empty.

## GetWindowsDesktops

Returns all the Windows desktop rights in the zone.

## Syntax

```
IWindowsDesktops GetWindowsDesktops()
```



## Return value

The collection of Windows desktop rights in the zone.

## PrecreateComputerZone

Adds a computer zone to a computer object in this zone.

## Syntax

```
IHierarchicalZoneComputer PrecreateComputerZone(string dnsname,  
DirectoryEntry trustee)
```

```
IHierarchicalZoneComputer PrecreateComputerZone(string dnsname,  
string trusteeDn)
```

## Parameters

Specify the following parameters when using this method.

Parameter	Description
<code>dnsname</code>	The DNS host name of the Active Directory computer object to which you wish to add a computer zone.
<code>trustee</code>	The user or group to which the computer-level overrides will be assigned.
<code>trusteeDn</code>	The user or group to which the computer-level overrides will be assigned, specified as a distinguished name.

## Return value

The hierarchical computer object that contains the computer zone.

## Discussion

Computer-level overrides for user, group, or computer role assignments are contained in a computer zone, which is a special type of zone that contains properties that are specific to only one computer. Computer zones are an internal data structure that are not exposed as zone in Access Manager.

This method adds a computer zone to a computer object in a hierarchical zone. You can then assign roles to that trustee.

Use `PrecreateComputerZone(string dnsname, DirectoryEntry trustee)` for .NET programs and `PrecreateComputerZone(string dnsname, string trusteeDn)` for VBScript.



## Exceptions

PrecreateComputerZone throws an `ApplicationException` if the method fails to delegate computer zone permissions (see the message returned by the exception for the reason).

## SetNSSVariable

Sets the value of the specified NSS environment variable; VBScript only.

## Syntax

```
string SetNssVariable (string name, string value)
```

## Parameter

Specify the following parameters when using this method.

Parameter	Description
name	The name of the variable.
value	The value of the variable.

## GroupDefaultName

Gets or sets the default name for new group profiles.

## Syntax

```
string GroupDefaultName {get; set;}
```

## Property value

The default group profile name.

## IsChild

Indicates whether this is a child zone.

## Syntax

```
bool IsChild {get;}
```



## Property value

Returns `true` if this zone is designated a child zone.

## Discussion

Centrify allows a child zone to have no parent, so that you can preload all the child zone profiles and role assignments before assigning the zone to a parent zone. This property identifies a zone as a parent zone if it has no link to a parent zone and has a child zone pointing to it. If the zone is not a parent zone according to those criteria, then this property returns `true`, identifying it as a child zone.

## IsGroupDefaultNameDefined

Determines whether the group default name has been defined.

## Syntax

```
bool IsGroupDefaultNameDefined {get; set;}
```

## Property value

Returns `true` if a group default name has been defined. Set this value `false` to clear this property. Call the `GroupDefaultName` property to set a value for this property.

## Exceptions

`IsGroupDefaultNameDefined` throws an `InvalidOperationException` if the group default name has not been defined and you attempt to set this property `true`.

## IsNextGidDefined

Determines whether the `NextGID` property has been defined for this zone.

## Syntax

```
bool IsNextGidDefined {get; set;}
```



## Property value

Returns true if the `NextGID` property has been defined. Set this value false to clear this property. Call the `NextGID` property to set a value for this property.

## Exceptions

`IsNextGidDefined` throws an `InvalidOperationException` if the `NextGID` property has not been defined and you attempt to set this property true.

## IsNextUidDefined

Determines whether the `NextUID` property has been defined for this zone.

## Syntax

```
bool IsNextUidDefined {get; set;}
```

## Property value

Returns true if the `NextUID` property has been defined. Set this value false to clear this property. Call the `NextUID` property to set a value for this property.

## Exceptions

`IsNextUidDefined` throws an `InvalidOperationException` if the `NextUID` property has not been defined and you attempt to set this property true.

## IsUseAutoPrivateGroupDefined

Determines whether the `UseAutoPrivateGroup` property has been defined for this zone.

## Syntax

```
bool IsUseAutoPrivateGroupDefined {get; set;}
```

## Property value

Returns true if the `UseAutoPrivateGroup` property has been defined. Set this value false to clear this property. Call the `UseAutoPrivateGroup` property to



set a value for this property.

## Discussion

When auto private groups are enabled, the user's UNIX profile name is automatically used as the group name and the user's UID is used as the GID.

## Exceptions

`IsUseAutoPrivateGroupDefined` throws an `InvalidOperationException` if the `UseAutoPrivateGroup` property has not been defined and you attempt to set this property true.

## IsUserDefaultGecosDefined

Determines whether the `UserDefaultGecos` property has been defined for this zone.

## Syntax

```
bool IsUserDefaultGecosDefined {get; set;}
```

## Property value

Returns true if the `UserDefaultGecos` property has been defined. Set this value false to clear this property. Call the `UserDefaultGecos` property to set a value for this property.

## Exceptions

`IsUserDefaultGecosDefined` throws an `InvalidOperationException` if the `UserDefaultGecos` property has not been defined and you attempt to set this property true.

## IsUserDefaultHomeDirectoryDefined

Determines whether the `UserDefaultHomeDirectory` property is defined for this zone.

## Syntax

```
bool IsUserDefaultHomeDirectoryDefined {get; set;}
```



## Property value

Returns `true` if the `UserDefaultHomeDirectory` property is defined. Set this value `false` to clear this property. Call the `DefaultHomeDirectory` property to set a value for this property.

## Exceptions

`IsUserDefaultHomeDirectoryDefined` throws an `InvalidOperationException` if the `UserDefaultHomeDirectory` property has not been defined and you attempt to set this property `true`.

## IsUserDefaultNameDefined

Determines whether the `UserDefaultName` property is defined for this zone.

## Syntax

```
bool IsUserDefaultNameDefined {get; set;}
```

## Property value

Returns `true` if the user default name property is defined. Set this value `false` to clear this property. Call the `UserDefaultName` property to set a value for this property.

## Exceptions

`IsUserDefaultNameDefined` throws an `InvalidOperationException` if the `UserDefaultName` property has not been defined and you attempt to set this property `true`.

## IsUserDefaultPrimaryGroupDefined

Determines whether the `UserDefaultPrimaryGroup` property is defined for this zone.

## Syntax

```
bool IsUserDefaultPrimaryGroupDefined {get; set;}
```



## Property value

Returns `true` if the user default primary group property is defined. Set this value `false` to clear this property. Call the `UserDefaultPrimaryGroup` property to set a value for this property.

## Exceptions

`IsUserDefaultPrimaryGroupDefined` throws an `InvalidOperationException` if the `UserDefaultPrimaryGroup` property has not been defined and you attempt to set this property `true`.

## IsUserDefaultRoleDefined

Determines whether the `UserDefaultRole` property is defined for this zone.

## Syntax

```
bool IsUserDefaultRoleDefined {get; set;}
```

## Property value

Returns `true` if the user default role property is defined. Set this value `false` to clear this property. Call the `UserDefaultRole` property to set a value for this property.

## Exceptions

`IsUserDefaultRoleDefined` throws an `InvalidOperationException` if the `UserDefaultRole` property has not been defined and you attempt to set this property `true`.

## IsUserDefaultShellDefined

Determines whether the `UserDefaultShell` property is defined for this zone.

## Syntax

```
bool IsUserDefaultShellDefined {get; set;}
```



## Property value

Returns true if the user default shell property is defined. Set this value false to clear this property. Call the [DefaultShell](#) property to set a value for this property.

## Exceptions

`IsUserDefaultShellDefined` throws an `InvalidOperationException` if the `UserDefaultShell` property has not been defined and you attempt to set this property true.

## NssVariables

Gets all the NSS environment variables.

## Syntax

```
IDictionary<string, string> NssVariables {get;}
```

## Property value

A dictionary of key-value pairs that define all the profile variables.

## Discussion

The `NssVariables` property is available only for .NET-based programs; call [GetNssVariables](#) for VBScript.

## Example

The following code sample illustrates using the `NssVariables` property in a script:

```
...
IHierarchicalZone objParent =
cims.GetZoneByPath("cn=" + strParentZone + "," + strContainerDN)
as IHierarchicalZone;
if (objParent == null)
{
    Console.WriteLine("Parent zone " + strParentZone + " does not
exist.");
}
else
{
    IHierarchicalZone objZone = cims.CreateZone(objContainer,
strZone) as
```



```
        IHierarchicalZone;
// set the starting UID and GID for the zone
objZone.NextUID = 10000;
objZone.NextGID = 10000;
objZone.UseNextUid = true;
objZone.UseNextGid = true;
objZone.AvailableShells = new string[] { "/bin/bash",
"/bin/shell" };
objZone.DefaultShell = "%{shell}";
objZone.DefaultHomeDirectory = "%{home}/%{user}";
objZone.UserDefaultGecos = "%{u:description}";
objZone.Parent = objParent;
objZone.NssVariables.Add("shell", "/bin/bash" );
objZone.Commit();
}
...
```

## Parent

Gets or sets the parent of the current zone.

## Syntax

```
IHierarchicalZone Parent {get; set;}
```

## Property value

The parent zone.

## Discussion

Centrify allows a child zone to have no parent, so that you can preload all the child zone profiles and role assignments before assigning the zone to a parent zone. See also the discussion under the [IsChild](#) property.

SFU zones cannot be child zones. See [Data storage for Centrify zones](#) for information about different zone types.

## Exceptions

Parent throws an `ApplicationException` if you attempt to assign a parent to an SFU zone.

## Example

The following code sample illustrates using the Parent property in a script:

• • • • •

```
...
IHierarchicalZone objParent =
cims.GetZoneByPath("cn=" + strParentZone + "," + strContainerDN)
as IHierarchicalZone;
if (objParent == null)
{
    Console.WriteLine("Parent zone " + strParentZone + " does not
exist.");
}
else
{
    IHierarchicalZone objZone = cims.CreateZone(objContainer,
strZone) as
    IHierarchicalZone;
    // set the starting UID and GID for the zone
    objZone.NextUID = 10000;
    objZone.NextGID = 10000;
    objZone.UseNextUid = true;
    objZone.UseNextGid = true;
    objZone.AvailableShells = new string[] { "/bin/bash",
"/bin/shell" };
    objZone.DefaultShell = "%{shell}";
    objZone.DefaultHomeDirectory = "%{home}/%{user}";
    objZone.UserDefaultGecos = "%{u:description}";
    objZone.Parent = objParent;
    objZone.NssVariables.Add("shell", "/bin/bash" );
    objZone.Commit();
}
...

```

## UseAppleGid

Determines whether to use the Apple algorithm to automatically generate the GID when adding a group to the zone. The Apple algorithm uses the globally unique identifier (GUID) for the object to generate a unique numeric identifier for each group.

### Syntax

```
bool UseAppleGid {get; set;}
```

### Property value

If this value is set to `true`, Centrify uses the Apple algorithm to generate the numeric group identifier (GID) when adding a group.



## UseAppleUid

Determines whether to use the Apple algorithm to automatically generate the UID when adding a user to the zone. The Apple algorithm uses the globally unique identifier (GUID) for the object to generate a unique numeric identifier for each user.

### Syntax

```
bool UseAppleUid {get; set;}
```

### Property value

If this value is set to `true`, Centrify uses the Apple algorithm to generate the numeric user identifier (UID) when adding a user.

## UseAutoGid

Determines whether to use the Centrify algorithm to automatically generate the GUID when adding a group to the zone. An auto-generated GUID when adding a group to the zone. The Centrify algorithm uses the domain-wide security identifier (SID) and the relative identifier (RID) to generate a unique numeric identifier for each group.

### Syntax

```
bool UseAutoGid {get; set;}
```

### Property value

If this value is set to `true`, Centrify automatically generates a GUID from the domain-wide security identifier (SID) and the relative identifier (RID) when adding a group.

## UseAutoPrivateGroup

Determines whether to use a private group when adding a user to the zone.

### Syntax

```
bool UseAutoPrivateGroup {get; set;}
```



## Property value

When this value is `true`, Centrify uses a private group when adding a user to the zone.

## Discussion

A private group automatically sets the user's UNIX profile name as the user's primary group name and the user's UID as the user's primary group GID. Automatically-generated groups are not stored or managed in Active Directory.

Call the `IsUseAutoPrivateGroupDefined` property before attempting to get the value for this property.

## Exceptions

`UseAutoPrivateGroup` may throw one of the following exceptions:

- `FormatException` if the GID is not a number.
- `InvalidOperationException` if the `UseAutoPrivateGroup` property has not been defined.

## UseAutoUid

Determines whether to use the Centrify algorithm to automatically generate the UID when adding a user to the zone. The Centrify algorithm uses the domain-wide security identifier (SID) and the relative identifier (RID) to generate a unique numeric identifier for each user.

## Syntax

```
bool UseAutoUid {get; set;}
```

## Property value

If this value is set to `true`, Centrify automatically generates a UID from the domain-wide security identifier (SID) and the relative identifier (RID) when adding a user.



## UseNextGid

Determines whether to automatically increment the GID when adding a group to the zone.

### Syntax

```
bool UseNextGid {get; set;}
```

### Property value

When this value is `true`, Centrify automatically increments the GID when adding a group to the zone.

### Discussion

There is no guarantee that the GIDs generated using this method are unique with regard to GIDs in other zones.

### Example

For a code sample illustrating the use of this property, see [Parent](#).

## UseNextUid

Determines whether to automatically increment the UID when adding a user to the zone.

### Syntax

```
bool UseNextUid {get; set;}
```

### Property value

When this value is `true`, Centrify automatically increments the UID when adding a user to the zone.

### Discussion

There is no guarantee that the UIDs generated using this method are unique with regard to UIDs in other zones.

• • • • •

## Example

For a code sample illustrating the use of this property, see [Parent](#).

## UserDefaultGecos

Gets or sets the default GECOS field for new user profiles in the zone.

### Syntax

```
string UserDefaultGecos {get; set;}
```

### Property value

The contents of the GECOS field.

## Example

For a code sample illustrating the use of this property, see [Parent](#).

## UserDefaultGid

Gets or sets the default GID (32-bit) for new user profiles in the zone.

### Syntax

```
int UserDefaultGid {get; set;}
```

### Property value

The GID. If set to -1, Centrify uses auto private group for new user profiles (see [UseAutoPrivateGroup](#)).

### Discussion

This property returns an error if the default GID has not been defined. Call the [IsUserDefaultPrimaryGroupDefined](#) property to determine whether the default GID is defined.

This property uses a 32-bit value for use in VBScript scripts. Use the [UserDefaultPrimaryGroup](#) property for 64-bit GIDs.



## Exceptions

UserDefaultGid may throw one of the following exceptions:

- `InvalidOperationException` if you try to get the default GID and it has not been defined.
- `FormatException` if the default GID value is not valid.

## UserDefaultName

Gets or sets the default name for new user profiles in the zone.

### Syntax

```
string UserDefaultName {get; set;}
```

### Property value

The default name.

## UserDefaultPrimaryGroup

Gets or sets the default GID (64-bit) for new user profiles in the zone.

### Syntax

```
long UserDefaultPrimaryGroup {get; set;}
```

### Property value

The GID. If set to -1, Centrify uses private groups for new user profiles (see `UseAutoPrivateGroup`).

### Discussion

This property returns an error if the default GID has not been defined. Call the `IsUserDefaultPrimaryGroupDefined` property to determine whether the default GID is defined.

This property uses a 64-bit value for use in .NET modules. Use the `UserDefaultGid` property for VBScript.



## Exceptions

UserDefaultPrimaryGroup may throw one of the following exceptions:

- `InvalidOperationException` if you try to get the default GID and it has not been defined.
- `FormatException` if the default GID value is not valid.

## UserDefaultRole

Gets or sets the default role for new user profiles in the zone.

### Syntax

```
IRole UserDefaultRole {get; set;}
```

### Property value

The default role.

## HierarchicalZoneComputer

The `HierarchicalZoneComputer` class represents a computer joined to a hierarchical zone.

### Syntax

```
public interface IHierarchicalZoneComputer : IComputer
```

### Discussion

The `HierarchicalZoneComputer` class inherits many methods and properties from the `Computer` class, but adds support for partial profiles and inheritable roles. Under hierarchical zones, both identity (profile data) and access (authorization data) are inherited, such that a computer's effective identity or access are determined by all the profile data and all the access data at all levels of the hierarchy.

See [HierarchicalUser](#) for a discussion of profile and access inheritance.



When you assign computer-level overrides for user, group, or computer role assignments, Centrify creates a computer zone, which is a special type of zone that contains the users, groups, and computer role assignments that are specific to only that one computer. Computer zones are not exposed as zones in Access Manager, but are referred to in the method and property descriptions where appropriate.

## Methods

The `HierarchicalZoneComputer` class provides the following methods:

This method	Does this
<code>AddAccessGroup</code>	Adds a group to the computer.
<code>AddGroupPartialProfile</code>	Adds a computer-specific partial profile for a specified group.
<code>AddLocalGroupPartialProfile</code>	Adds a computer-specific partial profile for a specified local group.
<code>AddLocalUserPartialProfile</code>	Adds a computer-specific partial profile for a specified user.
<code>AddRoleAssignment</code>	Adds an empty role assignment.
<code>AddUserPartialProfile</code>	Adds a computer-specific partial profile for a specified user.
<code>Commit</code>	Commits changes to the group object to Active Directory.  (Inherited from <code>Computer</code> .)
<code>CreateImportPendingGroup</code>	Creates a pending imported group in this computer.
<code>CreateImportPendingUser</code>	Creates a pending imported user in this computer.
<code>Delete</code>	Deletes the computer profile from Active Directory.  (Inherited from <code>Computer</code> .)
<code>DeleteAllProfiles</code>	Deletes all computer-specific users and groups.
<code>DeleteZone</code>	Deletes the computer zone object if it exists.
<code>GetAccessGroup</code>	Returns a group given a role for the group.
<code>GetAccessGroups</code>	Returns an enumeration of groups in the computer object.
<code>GetDirectoryEntry</code>	Returns the Active Directory object for the computer.  (Inherited from <code>Computer</code> .)
<code>GetEffectiveUserUnixProfiles</code>	Returns an enumeration of effective users under this computer zone.



<b>This method</b>	<b>Does this</b>
<code>GetGroupUnixProfile</code>	Returns the UNIX group profile in this computer zone for the specified Active Directory group.
<code>GetGroupUnixProfileByDN</code>	Returns the UNIX group profile in this computer zone for the Active Directory group specified by distinguished name.
<code>GetGroupUnixProfileByName</code>	Returns the UNIX group profile in this computer zone for the Active Directory group specified by group name.
<code>GetGroupUnixProfiles</code>	Returns an enumeration of the UNIX groups in this computer zone.
<code>GetImportPendingGroup</code>	Returns the group with the specified ID pending import.
<code>GetImportPendingGroups</code>	Returns an enumeration of groups pending import to this computer zone.
<code>GetImportPendingUser</code>	Returns the user with the specified ID pending import.
<code>GetImportPendingUsers</code>	Returns an enumeration of users pending import to this computer zone.
<code>GetIPendingGroupID</code>	Returns the numeric identifier for the pending import group with the specified group name.
<code>GetLocalGroupUnixProfile</code>	Returns the local UNIX group profile for a specified group name in the zone.
<code>GetLocalUserUnixProfileByDN</code>	Returns a local group profile using the distinguished name (DN) of the profile.
<code>GetLocalGroupUnixProfileByGid (Int32)</code>	Returns the local group profile using the Group Identifier (GID). This method is exposed to the .COM interface.
<code>GetLocalGroupUnixProfiles</code>	Returns a list of the local group profiles in the zone.
<code>GetLocalUserUnixProfile</code>	Returns the local user profile using the specified user name.
<code>GetLocalUserUnixProfileByDN</code>	Returns the local user profile specified by the distinguished name (DN) of the profile.
<code>GetLocalUserUnixProfileByUid (Int32)</code>	Returns the local user profile using the User Identifier (UID). This method is exposed to the .COM interface
<code>GetLocalUserUnixProfiles</code>	Returns a list of the local user profiles in the zone.
<code>GetIPendingUserID</code>	Returns the numeric identifier for the pending import user with the specified user name.
<code>GetNssVariable</code>	VBScript interface to access NSS variables.
<code>GetNSSVariables</code>	VBScript interface to obtain all NSS variable names.



<b>This method</b>	<b>Does this</b>
<code>GetPrimaryUser</code>	Returns the primary profile for the specified user.
<code>GetRoleAssignment</code>	Returns the role assignment for the specified role and trustee.
<code>GetRoleAssignmentById</code>	Returns the role assignment for the specified GUID.
<code>GetRoleAssignments</code>	Returns the collection of role assignments in the computer.
<code>GetRoleAssignmentToAllADUsers</code>	Returns the role assignment given to all Active Directory users who have a specified role.
<code>GetRoleAssignmentToAllUnixUsers</code>	Returns the role assignment given to all UNIX users who have a specified role.
<code>GetSecondaryUsers</code>	Returns an enumeration of the secondary profiles for the specified user.
<code>GetUserProfiles</code>	Returns an enumeration of all the user profiles for the specified user.
<code>GetUserRoleAssignments</code>	Returns an enumeration of all the user role assignments in this computer zone.
<code>GetUserUnixProfile</code>	Returns the UNIX user profile in this computer zone for the specified user.
<code>GetUserUnixProfileByDN</code>	Returns the UNIX user profile in this computer zone for the user specified by distinguished name.
<code>GetUserUnixProfileByName</code>	Returns the UNIX user profile in this computer zone for the user specified by user name.
<code>GetUserUnixProfileByUid</code>	Returns the UNIX user profile in this computer zone for the user specified by UID.
<code>GetUserUnixProfiles</code>	Returns an enumeration of all the UNIX user profiles in this computer zone.
<code>GroupUnixProfileExists</code>	Indicates whether the group has a profile in this computer zone.
<code>LocalGroupUnixProfileExists</code>	Indicates whether a UNIX profile exists in the zone for the specified local group.
<code>LocalUserUnixProfileExists</code>	Indicates whether a UNIX profile exists in the zone for the specified local user.
<code>Refresh</code>	Refreshes the data in this object instance from the data stored in Active Directory.  (Inherited from <code>Computer</code> .)
<code>SetNSSVariable</code>	VBScript interface to set the values of NSS variables.
<code>UserUnixProfileExists</code>	Indicates whether the specified user has a profile in this computer zone.



## Properties

The `HierarchicalZoneComputer` class provides the following properties:

This property	Does this
<code>AdsiInterface</code>	Gets the IADs interface of the zone object in Active Directory. (Inherited from <code>Computer</code> .)
<code>ADSPATH</code>	Gets the LDAP path to the zone object. (Inherited from <code>Computer</code> .)
<code>AgentVersion</code>	Gets the Active Directory client version number. (Inherited from <code>Computer</code> .)
<code>CanonicalName</code>	Gets the canonical name of the computer object. (Inherited from <code>Computer</code> .)
<code>ComputerZoneADSPATH</code>	Gets the LDAP path of the computer zone object.
<code>IsOrphan</code>	Indicates whether the CIMS data associated with this object is orphaned by the current credentials. (Inherited from <code>Computer</code> .)
<code>IsOrphanZone</code>	Indicates whether this computer is an orphan zone object.
<code>IsReadable</code>	Indicates whether the CIMS data associated with this object is readable with the current user credentials. (Inherited from <code>Computer</code> .)
<code>IsWritable</code>	Indicates whether the CIMS data associated with this object is writable with the current user credentials. (Inherited from <code>Computer</code> .)
<code>JBossEnabled</code>	Determines whether the computer is enabled for JBoss. (Inherited from <code>Computer</code> .)
<code>Name</code>	Gets or sets the name of the computer object. (Inherited from <code>Computer</code> .)
<code>NssVariables</code>	Gets the map of profile variables.
<code>ProfileADSPATH</code>	Gets the LDAP path to the computer UNIX profile. (Inherited from <code>Computer</code> .)
<code>SchemaVersion</code>	Gets the version of the data schema. (Inherited from <code>Computer</code> .)
<code>TomcatEnabled</code>	Determines whether the computer is enabled for Tomcat. (Inherited from <code>Computer</code> .)
<code>UserHomeDirectory</code>	Gets or sets the UNIX directory path that is used to substitute for <code>%{home}</code> in user profiles.



This property	Does this
<code>UserShell</code>	Gets or sets the shell that is used to substitute for <code>%{shell}</code> in user profiles.
<code>Version</code>	Gets the version number of the data schema. (Inherited from <code>Computer</code> .)
<code>WebLogicEnabled</code>	Determines whether the computer is enabled for WebLogic. (Inherited from <code>Computer</code> .)
<code>WebSphereEnabled</code>	Determines whether the computer is enabled for WebSphere. (Inherited from <code>Computer</code> .)
<code>Zone</code>	Gets or sets the zone that this computer joins.
<code>ZoneMode</code>	Gets the zone mode of the computer. (Inherited from <code>Computer</code> .)

## AddAccessGroup

Adds a group to the computer.

### Syntax

```
IMzRoleAssignment AddAccessGroup(DirectoryEntry groupDE)
```

```
IMzRoleAssignment AddAccessGroup(SearchResult groupSR)
```

```
IMzRoleAssignment AddAccessGroup(string groupDn)
```

```
IMzRoleAssignment AddAccessGroup(IAdsGroup groupIAds)
```

### Parameters

Specify one of the following parameters when using this method.

Parameter	Description
<code>groupDE</code>	The directory entry for the group you want to add.
<code>groupSR</code>	The directory entry for a group specified as a search result.
<code>groupDn</code>	The group specified as a distinguished name.
<code>groupIAds</code>	The IADs interface to the group.

### Return value

The computer role assignment that includes the specified group (`IMzRoleAssignment.TrusteeType==Group`).



## Discussion

The role assignment is not stored in Active Directory until you call the `Commit` method.

The `AddAccessGroup(DirectoryEntry groupDE)` and `AddAccessGroup(SearchResult groupSr)` methods are available only for .NET-based programs. Call `AddRoleAssignment` for VBScript.

## Exceptions

`AddAccessGroup` may throw one of the following exceptions:

- `ApplicationException` if the specified parameter is not a group or the method cannot find the group.
- `ArgumentNullException` if you pass a null parameter.

## AddGroupPartialProfile

Adds a computer-specific partial profile for the specified group to the computer.

## Syntax

```
IHierarchicalGroup AddGroupPartialProfile(DirectoryEntry groupDE)
```

```
IHierarchicalGroup AddGroupPartialProfile(SearchResult groupSR)
```

```
IHierarchicalGroup AddGroupPartialProfile(string groupDn)
```

```
IHierarchicalGroup AddGroupPartialProfile(IAdsGroup groupIAds)
```

## Parameters

Specify one of the following parameters when using this method.

Parameter	Description
<code>groupDE</code>	The directory entry for the group for which you want a partial profile.
<code>groupSr</code>	The directory entry for a group specified as a search result.
<code>groupDn</code>	The group specified as a distinguished name.
<code>groupIads</code>	The IADs interface to the group.

## Return value

The hierarchical group object that represents the group profile.



## Discussion

When you assign computer-level overrides for user, group, or computer role assignments, Centrify creates a computer zone, which is a special type of zone that contains the users, groups, and computer role assignments that are specific to only that one computer. Computer zones are not exposed as zones in Access Manager.

This method creates a computer zone and a new group profile with values set for the `Cims` and `user` properties. You can then add other properties to the profile.

The profile is not stored in Active Directory until you call the `Commit` method.

The `AddGroupPartialProfile(DirectoryEntry groupDE)` and `AddGroupPartialProfile(SearchResult groupSr)` methods are available only for .NET-based programs.

## Exceptions

If you pass a `null` or empty parameter, `AddGroupPartialProfile` throws the exception `ArgumentNullException`.

## Example

The `HierarchicalZoneComputer.AddGroupPartialProfile` method is used in the same way as the `HierarchicalZone.AddGroupPartialProfile` method. See [AddGroupPartialProfile](#) for an example.

## AddLocalGroupPartialProfile

Adds a partial profile for the specified group to the zone.

### Syntax

```
IHierarchicalUser AddLocalGroupPartialProfile(string groupName)
```

### Parameters

Specify `groupName`; the name of the local group.

### Return value

The hierarchical group object that represents the local group profile.



## Exceptions

If you pass a `null` parameter, `AddLocalGroupPartialProfile` throws the exception `ArgumentNullException`.

## AddLocalUserPartialProfile

Adds a partial profile for the specified user to the zone.

### Syntax

```
IHierarchicalUser AddLocalUserPartialProfile(string userName)
```

### Parameters

Specify `userName`; the user name of the local user.

### Return value

The hierarchical user object that represents the local user profile.

## Exceptions

If you pass a `null` parameter, `AddLocalUserPartialProfile` throws the exception `ArgumentNullException`.

## AddRoleAssignment

Adds an empty role assignment to the computer.

### Syntax

```
IRoleAssignment AddRoleAssignment()
```

### Return value

An empty role assignment object. This role assignment is not stored in Active Directory until you call the `RoleAssignment.Commit` method.



## AddUserPartialProfile

Adds a computer-specific partial profile for the specified user to the computer.

### Syntax

```
IHierarchicalUser AddUserPartialProfile(DirectoryEntry userDE)
```

```
IHierarchicalUser AddUserPartialProfile(SearchResult userSR)
```

```
IHierarchicalUser AddUserPartialProfile(string userDn)
```

```
IHierarchicalUser AddUserPartialProfile(IAdsUser userIAds)
```

### Parameters

Specify one of the following parameters when using this method.

Parameter	Description
userDE	The directory entry for the user for which you want a partial profile.
userSR	The directory entry for a user specified as a search result.
userDn	The user specified as a distinguished name.
userIads	The IADs interface to the user.

### Return value

The hierarchical user object that represents the user profile.

### Discussion

When you assign computer-level overrides for user, group, or computer role assignments, Centrify creates a computer zone, which is a special type of zone that contains the users, groups, and computer role assignments that are specific to only that one computer. Computer zones are not exposed as zones in Access Manager.

This method creates a computer zone and a new user profile with values set for the `Cims` and `User` properties. You can then add other properties to the profile.

The profile is not stored in Active Directory until you call the `Commit` method.

The `AddUserPartialProfile(DirectoryEntry userDE)` and `AddUserPartialProfile(SearchResult userSR)` methods are available only for .NET-based programs.



## Exceptions

If you pass a null or empty parameter, `AddUserPartialProfile` throws the exception `ArgumentNullException`.

## Example

The `HierarchicalZoneComputer.AddUserPartialProfile` method is used in the same way as the `HierarchicalZone.AddUserPartialProfile` method. See [AddUserPartialProfile](#) for an example.

## CreateImportPendingGroup

Creates a “pending import” group in this computer.

## Syntax

```
IGroupInfo CreateImportPendingGroup (string source, DateTime timestamp)
```

## Parameters

Specify the following parameters when using this method.

Parameter	Description
<code>source</code>	The location of the source data for the group to be imported.
<code>timestamp</code>	The date and time at which the data was retrieved.

## Return value

The newly created pending import group object.

## Discussion

Group profiles in a pending import group object needed to be mapped to Active Directory groups before they can be used. Groups in this state are normally imported from NIS domains or from text files and stored temporarily either in Active Directory or XML files until they are mapped to Active Directory accounts. For more information about importing and mapping groups, see the *Administrator’s Guide for Linux and UNIX*.



## CreateImportPendingUser

Creates a “pending import” user in this computer.

### Syntax

```
IUserInfo CreateImportPendingUser(string source, DateTime timestamp)
```

### Parameters

Specify the following parameters when using this method.

Parameter	Description
source	The location of the source data for the user to be imported.
timestamp	The date and time at which the data was retrieved.

### Return value

The newly created pending import user object.

### Discussion

User profiles in a pending import user object need to be mapped to Active Directory groups before they can be used. Users in this state are normally imported from NIS domains or from text files and stored temporarily either in Active Directory or in XML files until they are mapped to Active Directory accounts. For more information about importing and mapping users, see the *Administrator’s Guide for Linux and UNIX*.

## DeleteAllProfiles

Deletes all computer-specific users and groups.

### Syntax

```
void DeleteAllProfiles ()
```

### Discussion

Deleting a computer (**De**lete) doesn’t delete all the profiles associated with the computer. Use this method to delete computer-specific profiles after



deleting the computer.

## DeleteZone

Deletes the computer zone object.

### Syntax

```
void DeleteZone ()
```

### Discussion

When you assign computer-level overrides for user, group, or computer role assignments, Centrify creates a computer zone, which is a special type of zone that contains the users, groups, and computer role assignments that are specific to only that one computer. Computer zones are not exposed as zones in Access Manager.

This method deletes only the computer zone object (if it exists). Call the `Delete` method to delete the computer profile object.

## GetAccessGroup

Gets a user group assigned to this computer given a specific role.

### Syntax

```
IMzRoleAssignment GetAccessGroup(IRole role, DirectoryEntry  
group)
```

```
IMzRoleAssignment GetAccessGroup(IRole role, SearchResult  
groupSr)
```

```
IMzRoleAssignment GetAccessGroup(IRole role, string groupDn)
```

```
IMzRoleAssignment GetAccessGroup(IRole role, IADsGroup groupIAds)
```

### Parameters

Specify the following parameter when using this method:

Parameter	Description
<code>role</code>	The role of the group.

Specify one of the following parameters when using this method.



Parameter	Description
group	The directory entry for the group.
groupSr	The directory entry for the group specified as a search result.
groupDn	The group specified as a distinguished name.
groupIads	The IADs interface to the group.

## Return value

The computer role assignment that includes the specified group (`IMzRoleAssignment.TrusteeType==Group`).

## Discussion

Any number of user groups can be assigned to a computer role and each of those groups can have more than one role. Use this method to get the computer role assignment for a specific group and role.

The `GetAccessGroup(IRole role, DirectoryEntry groupDE)` and `GetAccessGroup(IRole role, SearchResult groupSr)` methods are available only for .NET-based programs. Call `GetRoleAssignment` for VBScript.

## Exceptions

`GetAccessGroup` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `ApplicationException` if the parameter value is not a valid user; or if it failed to create a role assignment because it cannot find the user.

## Example

The `HierarchicalZoneComputer.GetAccessGroup` method is used in the same way as the `HierarchicalZone.GetAccessGroup` method. See [AddAccessGroup](#) for an example of using the `HierarchicalZone.GetAccessGroup` method in a script:

## GetAccessGroups

Returns the computer roles assigned to this computer.



## Syntax

```
IRoleAssignments GetAccessGroups()
```

## Return value

The collection of computer roles. Enumerate this object to get all of the `IAzRoleAssignment` objects for this computer.

## GetEffectiveUserUnixProfiles

Returns the collection of effective users under this computer zone.

## Syntax

```
IUserUnixProfiles GetEffectiveUserUnixProfiles()
```

## Return value

A collection of `IHierarchicalUser` objects representing all the user profiles under this computer zone, including those inherited from zones higher in the hierarchy.

## GetGroupUnixProfile

Returns the hierarchical group profile for a specified Active Directory group in the computer zone.

## Syntax

```
IHierarchicalGroup GetGroupUnixProfile(IGroup group)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
group	The group for which you want profile information.

## Return value

The profile for the specified group, or `null` if none is found.



## Discussion

This method uses the `Centrify.DirectControl.API.IGroup` group returned by a `Cims.GetGroup` or `Cims.GetGroupByPath` call to retrieve the group profile.

## Exceptions

`GetGroupUnixProfile` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `NotSupportedException` if the computer zone schema is not supported.
- `ApplicationException` if there is more than one group with the specified `IGroup` value in the zone.

## GetGroupUnixProfileByDN

Returns the UNIX profile for a group in this computer zone using the distinguished name (DN) of the profile.

## Syntax

```
IHierarchicalGroup GetGroupUnixProfileByDN(string dn)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>dn</code>	The distinguished name (DN) of the group profile.

## Return value

The group profile with the distinguished name (DN) specified, or `null` if no matching group profile is found.

## Discussion

The group profile is the service connection point associated with the Active Directory group object.



## Exceptions

`GetGroupUnixProfileByDN` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `NotSupportedException` if the computer zone schema is not supported.

## GetGroupUnixProfileByName

Returns the hierarchical group profile for a group with the specified name in the computer zone.

### Syntax

```
IHierarchicalGroup GetGroupUnixProfileByName(string name)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>name</code>	The name of the UNIX group profile for which you want to retrieve information.

### Return value

The profile for the specified group name, or `null` if no profile is found.

### Discussion

The name you specify should be the UNIX group name for the group if it differs from the Active Directory name for the group.

## Exceptions

`GetGroupUnixProfileByName` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null` or empty.
- `NotSupportedException` if the computer zone schema is not supported.
- `ApplicationException` if there is more than one group with the specified name in the zone.



## GetGroupUnixProfiles

Returns the collection of UNIX group profiles that have been defined for the computer zone.

### Syntax

```
IGroupUnixProfiles GetGroupUnixProfiles()
```

### Return value

Returns a collection of `GroupUnixProfile` objects.

## GetImportPendingGroup

Returns a group pending import to the computer zone given the GUID.

### Syntax

```
IGroupInfo GetImportPendingGroup(string id, bool storePendingAD, string storePendingFilePath)
```

### Parameter

Specify the following parameters when using this method:

Parameter	Description
<code>id</code>	The GUID of the group that's pending import.
<code>storePendingAD</code>	Specify <code>true</code> if the group is being imported from Active Directory. Specify <code>false</code> if the group is being imported from an XML file.
<code>storePendingFilePath</code>	The file path of the XML file.

### Return value

The `IGroupInfo` object for the specified group.

### Discussion

This method takes the `storePendingAD` Boolean and the `storePendingFilePath` information, stores them in the group profile, then finds and returns the group pending import that has the specified GUID.



Group profiles that are pending import are normally imported from NIS domains or from text files and not yet mapped to Active Directory groups. For more information about importing and mapping groups, see the *Administrator's Guide for Linux and UNIX*.

## GetImportPendingGroups

Returns the list of groups pending import to this computer zone.

### Syntax

```
IGroupInfos GetImportPendingGroups()
```

### Return value

The collection of group profiles pending import to this computer zone.

## GetImportPendingUser

Returns an individual user pending import for this computer given the GUID.

### Syntax

```
IUserInfo GetImportPendingUser(string id, bool storePendingAD, string storePendingFilePath)
```

### Parameter

Specify the following parameters when using this method:

Parameter	Description
id	The GUID of the user that's pending import.
storePendingAD	Specify <code>true</code> if the user is being imported from Active Directory. Specify <code>false</code> if the user is being imported from an XML file.
storePendingFilePath	The file path of the XML file.

### Return value

The `IUserInfo` object for the specified ID in the computer.



## Discussion

This method takes the `storePendingAD` Boolean and the `storePendingFilePath` information, stores them in the user profile, then finds and returns the user pending import that has the specified GUID.

User profiles that are pending import are normally imported from NIS domains or from text files and not yet mapped to Active Directory groups. For more information about importing and mapping groups, see the *Administrator's Guide for Linux and UNIX*.

## GetImportPendingUsers

Returns the collection of users pending import to this computer zone.

### Syntax

```
IUserInfos GetImportPendingUsers()
```

### Return value

The collection of user profiles pending import to this computer zone.

## GetIPendingGroupID

Returns the numeric identifier of the pending import group.

### Syntax

```
IGroupInfo GetPendingGroupID()
```

### Return value

The pending import group specified by group name to the selected zone.

## GetIPendingUserID

Returns the numeric identifier of the pending import user.

• • • • •

## Syntax

```
IUserInfo GetPendingUserID()
```

## Return value

The pending import user specified by the user name to the selected zone.

## GetLocalGroupUnixProfile

Returns the UNIX group profile for a specified local group.

## Syntax

```
IGroupUnixProfile GetLocalGroupUnixProfile(string groupName)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
groupName	The name of the local group for which you want to retrieve profile information.

## Return value

The `GroupUnixProfile` object for the specified local group name. If there is no group, `null` is returned.

## Exceptions

`GetGroupUnixProfile` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.

## GetLocalGroupUnixProfileByDN

Returns the Local UNIX profile for a group in the zone using the distinguished name (DN) of the profile.

## Syntax

```
IGroupUnixProfile GetLocalGroupUnixProfileByDN(string dn)
```



## Parameter

Specify the following parameters when using this method.

Parameter	Description
dn	The distinguished name (DN) of the local group profile.

## Return value

The local group profile with the distinguished name (DN) matching the distinguished name specified, or null if no matching group profile is found.

## GetLocalGroupUnixProfileByGid (Int32)

Returns the Local UNIX profile for a group in the zone using the group identifier (GID) of the profile. This method is exposed to the .COM interface.

## Syntax

```
IGroupUnixProfile GetLocalGroupUnixProfileByGid(int gid)
```

## Parameter

Specify the following parameters when using this method.

Parameter	Description
gid	The group identifier (GID) of the local group profile.

## Return value

The local group profile with the specified group identifier (GID) or null if no matching group profile is found.

## GetLocalGroupUnixProfiles

Get the list of local group profiles in the zone.

## Syntax

```
IGroupUnixProfiles GetLocalGroupUnixProfiles()
```



## Return value

Returns a collection of GroupUnixProfile objects. If there are no groups, null is returned.

## GetLocalUserUnixProfile

Returns the UNIX user profile for a specified local group.

## Syntax

```
IUserUnixProfile GetLocalUserUnixProfile(string userName)
```

## Parameter

Specify the userName parameter when using this method.

## Return value

Returns the local user profile with the specified user name. If there is no group, null is returned.

## GetLocalUserUnixProfileByDN

Returns the local UNIX profile for a user in the zone using the distinguished name (DN) of the profile.

## Syntax

```
IUserUnixProfile GetLocalUserUnixProfileByDN(string dn)
```

## Parameter

Specify the following parameters when using this method.

Parameter	Description
dn	The distinguished name (DN) of the local user profile.



## GetLocalUserUnixProfileByUid (Int32)

Returns the local UNIX profile for a user in the zone using the user identifier (GID) of the profile. This method is exposed to the .COM interface.

### Syntax

```
IUserUnixProfile GetLocalUserUnixProfileByUid(int uid)
```

### Parameter

Specify the following parameters when using this method.

Parameter	Description
uid	The user identifier (UID) of the local user profile.

### Return value

The local user profile with the specified user identifier (UID) or null if no matching user profile is found.

## GetLocalUserUnixProfiles

Get a list of local UNIX user profiles in the zone.

### Syntax

```
IUserUnixProfiles GetLocalUserUnixProfiles()
```

### Return value

Returns a collection of local user profiles in the zone. If there are no users, null is returned.

## GetNssVariable

Returns the specified NSS environment variable; VBScript only.

### Syntax

```
string GetNssVariable (string name)
```



## Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the variable.

## Return value

The value of the variable, or null if there is no NSS variable with the specified name.

## GetNSSVariables

Returns the names of all NSS variables; VBScript only.

## Syntax

```
IEnumerable GetNSSVariables()
```

## Return value

Returns a collection of NSS variable names.

## GetPrimaryUser

Returns the primary profile for the specified user.

## Syntax

```
IHierarchicalUser GetPrimaryUser(DirectoryEntry userDE)
```

```
IHierarchicalUser GetPrimaryUser(SearchResult userSR)
```

```
IHierarchicalUser GetPrimaryUser(string userDn)
```

```
IHierarchicalUser GetPrimaryUser(IAdsUser userIAds)
```

## Parameters

Specify one of the following parameters when using this method.



Parameter	Description
userDE	The directory entry for the user for which you want the primary profile.
userSr	The directory entry for a user specified as a search result.
userDn	The user specified as a distinguished name.
userIads	The IADs interface to the user.

## Return value

The hierarchical user object that represents the user profile.

## Discussion

The primary profile is the profile at the highest level in the zone hierarchy where the user's profile is defined. All or part of the primary profile can be overridden by secondary profiles farther down in the hierarchy.

The `GetPrimaryUser(DirectoryEntry userDE)` and `GetPrimaryUser(SearchResult userSr)` methods are available only for .NET-based programs.

## Exceptions

`GetPrimaryUser` throws an `ArgumentNullException` if the specified parameter value is `null` or empty.

## GetRoleAssignment

Returns a role assignment given a role and trustee.

## Syntax

```
IRoleAssignment GetRoleAssignment (IRole role, string dn)
```

## Parameter

Specify the following parameters when using this method.

Parameter	Description
role	The role for which you want the assignment.
dn	The distinguished name of the user or group to whom the role is assigned.



## Return value

The role assignment, or null if no match is found.

## Exceptions

GetRoleAssignment throws an `ArgumentNullException` if either specified parameter value is null or empty.

## GetRoleAssignmentById

Returns a role assignment given an ID.

## Syntax

```
IRoleAssignment GetRoleAssignment (Guid id)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
id	The GUID of the role assignment.

## Return value

The role assignment, or null if no match is found.

## Exceptions

GetRoleAssignmentById throws an `ArgumentNullException` if the specified parameter value is empty.

## GetRoleAssignments

Returns all the role assignments in the computer.

## Syntax

```
IRoleAssignments GetRoleAssignments()
```



## Return value

The collection of role assignments in the computer.

## GetRoleAssignmentToAllADUsers

Returns the role assignment given to all Active Directory users who have a specified role.

### Syntax

```
IRoleAssignment GetRoleAssignmentToAllADUsers(IRole role)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
role	The user role for which you want the role assignment.

## Return value

The role assignment for the specified role.

### Exceptions

`GetRoleAssignmentToAllADUsers` throws an `ArgumentNullException` if the specified parameter value is `null`.

## GetRoleAssignmentToAllUnixUsers

Returns the role assignment given to all UNIX users who have a specified role.

### Syntax

```
IRoleAssignment GetRoleAssignmentToAllUnixUsers(IRole role)
```

### Parameter

Specify the following parameter when using this method:



---

Parameter	Description
role	The user role for which you want the role assignment.

---

## Return value

The role assignment for the specified role.

## Discussion

This method returns the role assignment for all local UNIX users with the specified role.

## Exceptions

`GetRoleAssignmentToAllUnixUsers` throws an `ArgumentNullException` if the specified parameter value is `null`.

## GetSecondaryUsers

Returns the secondary profiles for the specified user.

## Syntax

```
IUserUnixProfiles GetSecondaryUsers(DirectoryEntry userDE)
```

```
IUserUnixProfiles GetSecondaryUsers(SearchResult usersSR)
```

```
IUserUnixProfiles GetSecondaryUsers(string userDn)
```

```
IUserUnixProfiles GetSecondaryUsers(IAdsUser userIAds)
```

## Parameters

Specify one of the following parameters when using this method.

---

Parameter	Description
userDE	The directory entry for the user for which you want the secondary profiles.
userSr	The directory entry for a user specified as a search result.
userDn	The user specified as a distinguished name.
userIads	The IADs interface to the user.

---

## Return value

The collection of secondary user UNIX profiles.



## Discussion

The primary profile is the profile at the highest level in the zone hierarchy where the user's profile is defined. All or part of the primary profile can be overridden by secondary profiles farther down in the hierarchy.

The `GetSecondaryUsers(DirectoryEntry userDE)` and `GetSecondaryUsers(SearchResult userSr)` methods are available only for .NET-based programs.

## Exceptions

`GetSecondaryUsers` throws an `ArgumentNullException` if the specified parameter value is `null` or the user does not exist.

## GetUserProfiles

Returns all the profiles for the specified user.

## Syntax

```
IUserUnixProfiles GetUserProfiles(DirectoryEntry userDE)
```

```
IUserUnixProfiles GetUserProfiles(SearchResult userSR)
```

```
IUserUnixProfiles GetUserProfiles(string userDn)
```

```
IUserUnixProfiles GetUserProfiles(IAdsUser userIAds)
```

## Parameters

Specify one of the following parameters when using this method.

Parameter	Description
<code>userDE</code>	The directory entry for the user for which you want the profiles.
<code>userSr</code>	The directory entry for a user specified as a search result.
<code>userDn</code>	The user specified as a distinguished name.
<code>userIads</code>	The IADs interface to the user.

## Return value

The collection of user UNIX profiles.



## Discussion

The `GetUserProfiles(DirectoryEntry userDE)` and `GetUserProfiles(SearchResult userSr)` methods are available only for .NET-based programs.

## Exceptions

`GetUserProfiles` throws an `ArgumentNullException` if the specified parameter value is `null` or the user does not exist.

## GetUserRoleAssignments

Returns all the user role assignments in the computer zone, or for a specific user in the computer zone.

## Syntax

```
IRoleAssignments GetUserRoleAssignments()  
IRoleAssignments GetUserRoleAssignments(DirectoryEntry userDE)  
IRoleAssignments GetUserRoleAssignments(SearchResult userSR)  
IRoleAssignments GetUserRoleAssignments(string userDn)  
IRoleAssignments GetUserRoleAssignments(IAdsUser userIAds)  
IRoleAssignments GetUserRoleAssignments(IUser user)
```

## Parameters

Specify no parameters to return all the role assignments in the computer zone.

Specify one of the following parameters to return all the role assignments for a specific user:

Parameter	Description
<code>userDE</code>	The directory entry for the user for which you want the role assignments.
<code>userSr</code>	The directory entry for a user specified as a search result.
<code>userDn</code>	The user specified as a distinguished name.
<code>userIads</code>	The IADs interface to the user.
<code>user</code>	The user specified as a CIMS user object.



## Return value

The collection of role assignments as `IRoleAssignment` objects.

## Discussion

The `GetUserRoleAssignments(DirectoryEntry userDE)` and `GetUserRoleAssignments(SearchResult userSr)` methods are available only for .NET-based programs.

## Exceptions

`GetUserRoleAssignments` throws an `ArgumentNullException` if the required parameter is null or empty.

## GetUserUnixProfile

Returns the UNIX user profile for a specified Active Directory user in this computer zone.

## Syntax

```
IHierarchicalUser GetUserUnixProfile(IUser user)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>user</code>	The user object for Active Directory user.

## Return value

The profile for the specified Active Directory user, or null if none could be found.

## Discussion

This method uses the `Centrifify.DirectControl.API.IUser` returned by a `GetUser` or `GetUserByPath` call to retrieve the user profile.



## Exceptions

`getUserUnixProfile` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `NotSupportedException` if the computer zone schema is not supported.

## GetUserUnixProfileByDN

Returns the UNIX profile for a user in this computer zone given the distinguished name (DN) of the profile.

### Syntax

```
IHierarchicalUser GetUserUnixProfileByDN(string dn)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>dn</code>	The distinguished name (DN) of the user profile.

### Return value

The user profile with the distinguished name (DN) specified, or `null` if no matching user profile is found.

### Discussion

The user profile is the service connection point associated with the Active Directory user object.

## Exceptions

`getUserUnixProfileByDN` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `NotSupportedException` if the computer zone schema is not supported.



## GetUserUnixProfileByName

Returns the UNIX profile for a user in this computer zone given the user name.

### Syntax

```
IHierarchicalUser GetUserUnixProfileByName(string name)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The user's UNIX login name.

### Return value

The profile of the specified user, or `null` if none is found.

### Exceptions

`GetUserUnixProfileByName` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null` or empty.
- `NotSupportedException` if the computer zone schema is not supported.
- `ApplicationException` if there is more than one user with the specified name in the zone.

## GetUserUnixProfileByUid

Returns the UNIX profile for a user in this computer zone given the user identifier (UID).

### Syntax

```
IHierarchicalUser GetUserUnixProfileByUid(int uid)  
IHierarchicalUser GetUserUnixProfileByUid(long uid)
```

### Parameter

Specify the following parameter when using this method:



Parameter	Description
<code>uid</code>	The user identifier (UID) associated with the Active Directory user.

## Return value

The user profile for the specified UID in the computer zone, or `null` if none is found.

## Discussion

If there are multiple user profiles with the UID specified, this method returns only the first user profile found. To find all user profiles with a specific UID, use the `GetUserUnixProfiles` method to return the collection of profiles for a computer, then search the collection for the UID.

**Note:** There are two versions of this method: one designed for COM-based programs that supports a 32-bit signed number for the `uid` argument and one designed for .NET-based programs that allows a 64-bit signed number for the `uid` argument. These methods are provided for backward compatibility with earlier versions of Centrify software. These methods are not applicable for version 4.0 or later.

## Exceptions

`GetUserUnixProfileByUid` may throw one of the following exceptions:

- `ArgumentException` if you specify a negative UID.
- `NotSupportedException` if the computer zone schema is not supported.

## GetUserUnixProfiles

Returns the list of UNIX user profiles in this computer zone.

## Syntax

```
IComputerUserUnixProfiles GetUserUnixProfiles()
```

## Return value

The collection of computer user UNIX profiles.



## GroupUnixProfileExists

Indicates whether a UNIX profile exists for the specified group in this computer zone.

### Syntax

```
bool GroupUnixProfileExists(IGroup group)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
group	The group for which you want to check whether a UNIX profile exists.

### Return value

Returns `true` if a UNIX profile is found in this computer zone for the specified group, or `false` if no UNIX profile exists for the group in the computer zone.

### Exceptions

`GetUserUnixProfileById` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `NotSupportedException` if the computer zone schema is not supported.

## LocalGroupUnixProfileExists

Checks whether a UNIX profile exists for the specified local group in the zone.

### Syntax

```
bool LocalGroupUnixProfileExists(string groupName)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
groupName	The group name for which you want to check whether a UNIX profile exists.



## Return value

Returns `true` if the local UNIX group profile is found in the zone, or `false` if no UNIX profile exists for the group in the zone.

## Exceptions

`LocalGroupUnixProfileExists` may throw the following exception:

- `ArgumentNullException` if the specified parameter value is `null`.

## LocalUserUnixProfileExists

Checks whether a local UNIX profile exists for the specified user in the zone.

## Syntax

```
bool LocalUserUnixProfileExists(string userName)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>userName</code>	The local user name for which you want to check whether a UNIX profile exists.

## Return value

Returns `true` if a UNIX profile is found in the zone for the specified local user, or `false` if no UNIX profile exists for the user in the zone.

## Exceptions

`UserUnixProfileExists` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.

## SetNSSVariable

Sets the value of the specified NSS environment variable; VBScript only.



## Syntax

`string SetNssVariable (string name, string value)`

## Parameter

Specify the following parameters when using this method.

Parameter	Description
<code>name</code>	The name of the variable.
<code>value</code>	The value of the variable. Pass <code>null</code> to remove the variable.

## UserUnixProfileExists

Indicates whether a UNIX profile exists for the specified user in this computer zone.

## Syntax

`bool UserUnixProfileExists(IUser user)`

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>user</code>	The user name for which you want to check whether a UNIX profile exists.

## Return value

Returns `true` if a UNIX profile is found for the specified user, or `false` if no UNIX profile exists for the user in the computer zone.

## Exceptions

`UserUnixProfileExists` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `NotSupportedException` if the computer zone schema is not supported.



## ComputerZoneADsPath

Gets the LDAP path of the computer zone object.

### Syntax

```
string ComputerZoneADsPath {get;}
```

### Property value

The LDAP path.

### Discussion

When you assign computer-level overrides for user, group, or computer role assignments, Centrify creates a *computer zone*, which is a special type of zone that contains the users, groups, and computer role assignments that are specific to only that one computer. Computer zones are not exposed as zones in Access Manager.

If the computer zone does not contain any users, groups, or computer roles, this property returns `null` or `string.Empty`.

## IsOrphanZone

Indicates whether this computer zone is an orphan zone.

### Syntax

```
bool OrphanZone {get;}
```

### Property value

Returns `true` if this computer zone is an orphan zone object.

### Discussion

The computer zone is an orphan if it has no corresponding service connection point (SCP) and Active Directory computer object.

When you assign computer-level overrides for user, group, or computer role assignments, Centrify creates a *computer zone*, which is a special type of zone that contains the users, groups, and computer role assignments that are



specific to only that one computer. Computer zones are not exposed as zones in Access Manager.

## NssVariables

Gets all the NSS environment variables.

### Syntax

```
IDictionary<string, string> NssVariables {get;}
```

### Property value

A dictionary of key-value pairs that define all the profile variables.

### Discussion

This property uses a 64-bit value for use in .NET modules. Use the `GetNssVariables` property for VBScript.

## UserHomeDirectory

Gets or sets the UNIX directory path that is used to substitute for `%{home}` in user profiles.

### Syntax

```
string UserHomeDirectory {get; set;}
```

### Property value

The UNIX directory path, or `null` to inherit the path from the parent zone.

## UserShell

Gets or sets the shell that is used to substitute for `%{shell}` in user profiles.

### Syntax

```
int UserDefaultGid {get; set;}
```



## Property value

The user shell, or null to inherit the shell from the parent zone.

## Zone

Determines the zone object for the zone to which the computer is currently joined.

## Syntax

```
IHierarchicalZone Zone {get; set;}
```

## Property value

The zone the computer account has joined.

## Discussion

Each computer object can only be associated with one zone: the zone used to join the computer to its Active Directory domain.

## Exceptions

Zone throws an `ApplicationException` if you try to set the zone to a type that is not hierarchical or is not supported or if the computer is already joined to a zone and you try set a new zone.

# HZRoleAssignment

Represents a zone-level role assignment.

## Syntax

```
public interface IRoleAssignment
```

## Methods

The `HZRoleAssignment` class provides the following methods:



This method	Does this
<code>ClearCustomAttributes</code>	VBScript interface to clear the custom attributes for this class. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>Commit</code>	Commits changes in the role assignment to Active Directory. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>Delete</code>	Deletes the role. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>ICustomAttributeContainer</code> <code>GetCustomAttributeContainer</code>	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>GetTrustee</code>	Returns the trustee being assigned. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>SetCustomAttribute</code>	VBScript interface to set the custom attributes for this class. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>Validate</code>	Validates the role assignment. Inherited from ( <a href="#">RoleAssignment.</a> )

## Properties

The `HZRoleAssignment` class provides the following properties:

This property	Does this
<code>CustomAttributes</code>	VBScript only: Gets or sets custom attributes for this class. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>EndTime</code>	Determines the time at which this role becomes inactive. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>Id</code>	Gets the GUID of the role assignment. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>IsRoleOrphaned</code>	Indicates whether the role assignment is orphaned due to a missing or invalid role. Inherited from ( <a href="#">RoleAssignment.</a> )
<code>IsTrusteeOrphaned</code>	Indicates whether the role assignment is orphaned due to a missing or invalid trustee.



This property	Does this
	Inherited from ( <a href="#">RoleAssignment.</a> )
<a href="#">LocalTrustee</a>	Gets the local trustee being assigned. Inherited from ( <a href="#">RoleAssignment.</a> )
<a href="#">Role</a>	Gets the role the trustee is assigned to. Inherited from ( <a href="#">RoleAssignment.</a> )
<a href="#">StartTime</a>	Specifies the time from which this role becomes effective. Inherited from ( <a href="#">RoleAssignment.</a> )
<a href="#">TrusteeDn</a>	Gets the distinguished name of the trustee assigned this role. Inherited from ( <a href="#">RoleAssignment.</a> )
<a href="#">TrusteeType</a>	Gets the trustee type of the role assignment. Inherited from ( <a href="#">RoleAssignment.</a> )
<a href="#">Zone</a>	Gets the zone in which the role assignment is made.

## Zone

Gets the zone in which the role assignment is made.

### Syntax

```
IZone Zone {get;}
```

### Property value

The zone of the role assignment.

## InheritedRoleAsg

The `InheritedRoleAsg` class represents a virtual role assignment constructed by inheritance from parent zones and the current zone or computer. A role assignment object contains information about an Active Directory object (trustee) that has been added to a role.

### Syntax

```
public interface IInheritedRoleAsg
```



## Methods

The `InheritedRoleAsg` class provides the following method:

This method	Does this
<code>GetTrustee</code>	Returns the user associated with the role.

## Properties

The `InheritedRoleAsg` class provides the following properties:

This property	Does this
<code>EndTime</code>	Determines the time at which this role becomes inactive.
<code>IsRoleOrphaned</code>	Indicates whether the role assignment is orphaned due to missing or invalid data.
<code>IsTrusteeOrphaned</code>	Indicates whether the role assignment is orphaned due to a missing trustee.
<code>Role</code>	Gets the role the trustee is assigned to.
<code>Source</code>	Gets the role assignment that is the source for this inherited role assignment.
<code>StartTime</code>	Specifies the time from which this role becomes effective.
<code>TrusteeDn</code>	Gets the distinguished name of the trustee assigned this role.

## GetTrustee

Returns the user associated with the role.

### Syntax

```
DirectoryEntry GetTrustee()
```

### Return value

The Active Directory object representing the user.

## EndTime

Determines the time from which this role becomes inactive.



## Syntax

```
DateTime EndTime {get; set;}
```

## Property value

The time at which the role ceases to be active. A value of `DateTime.MaxValue` means the role never expires. The time must be later than 1/1/1970 00:00.

## IsRoleOrphaned

Indicates whether the role assignment is orphaned due to a missing or invalid role.

## Syntax

```
bool IsRoleOrphaned {get;}
```

## Property value

Returns true if this role assignment is an orphan.

## IsTrusteeOrphaned

Indicates whether the role assignment is orphaned due to a missing or invalid user.

## Syntax

```
bool IsTrusteeOrphaned {get;}
```

## Property value

Returns true if this role assignment is an orphan.

## Role

## Syntax

```
IRole Role {get;}
```



## Property value

The object representing the role.

## Source

Gets the role assignment that is the source for this inherited role assignment.

## Syntax

```
IRoleAssignment Source {get;}
```

## Property value

The original role assignment that is the source for this inherited role assignment.

## StartTime

Gets the time from which this role becomes effective.

## Syntax

```
DateTime StartTime {get;}
```

## Property value

The time at which the role becomes active. A value of `DateTime.MinValue` means the role becomes effective immediately. The time must be later than 1/1/1970 00:00.

## TrusteeDn

Gets the user associated with the role.

## Syntax

```
string TrusteeDn {get;}
```



## Property value

The distinguished name of the user.

## Key

The key class provides access to individual license key properties.

## Syntax

```
public interface IKey
```

## Discussion

A key object represents a single license key provided by Centrify. The license key is an encrypted string that encapsulates information such as the license type, number of allowed computers, a serial number, and whether the license is an evaluation or permanent license. For example, a single license key might authorize access for 25 servers for the organization with the serial number defined as 317.

## Properties

The key class provides the following properties:

This property	Does this
<b>Count</b>	Gets the number of licenses provided by a specific key.
<b>ExpiryDate</b>	Gets the date a specific license key is set to expire.
<b>IsEval</b>	Determines whether the license key is an evaluation license.
<b>IsValid</b>	Determines whether the license key being checked is a valid license.
<b>SerialNumber</b>	Gets the serial number of the license key.
<b>Type</b>	Gets the license type for a specific license key.

## Count

Gets the number of licenses provided by a specific license key.



## Syntax

```
int Count {get;}
```

## Property value

The number of licenses provided in a license key.

## Discussion

Each license key specifies the number of workstations or servers for which you have purchased licenses. This property indicates the total number of licenses defined for the key. It does not indicate the number currently in use or available to be used.

## Example

The following code sample illustrates using the license key Count property in a script:

```
...
set objCollection cims.LoadLicenses
for each objLics in objCollection
    wscript.echo "Number of licenses:", objLics.Count
    for each objLic in objLics
        wscript.Echo "License Type:", objLic.Type
        wscript.Echo "Seats:", objLic.Count
        wscript.Echo "Used:", objLic.usedCount
        set objKeys = objLic.keys
        i = 0
        do while i < objKeys.Count
            set objKey = objKeys(0)
            if objKey.isEval then
                wscript.Echo "-- [Eval] ", objKey.ExpiryDate
            else
                wscript.Echo "--", "Serial #:",
objKey.SerialNumber
                wscript.Echo "Seats:", objKey.Count
            end if
            i = i + 1
        loop
    next
    wscript.Echo ""
next
...
```

## ExpiryDate

Gets the date a specific license key is set to expire.

• • • • •

## Syntax

```
DateTime ExpiryDate {get;}
```

## Property value

The date on which a specified license key expires.

## Discussion

If a license key is defined as an evaluation license, it includes a timestamp that determines when the license will expire. You can use this property to retrieve this expiration date.

## Example

The following code sample illustrates using `ExpiryDate` in a script:

```
...
set objCollection cims.LoadLicenses
for each objLics in objCollection
    wscript.echo "Number of licenses:", objLics.Count
    for each objLic in objLics
        wscript.Echo "License Type:", objLic.Type
        wscript.Echo "Seats:", objLic.Count
        wscript.Echo "Used:", objLic.usedCount
        'Display the expiration for eval licenses
        set objKeys = objLic.keys
        i = 0
        do while i < objKeys.Count
            set objKey = objKeys(0)
            if objKey.isEval then
                wscript.Echo "-- [Eval] ", objKey.ExpiryDate
            end if
            i = i + 1
        loop
    next
    wscript.Echo ""
next
...
```

## IsEval

Determines whether the license key is an evaluation license.

## Syntax

```
bool IsEval {get;}
```



## Property value

Returns `true` if the license is a temporary evaluation license, or `false` if the license key is a permanent license.

## Discussion

An evaluation license provides full use of Centrify software for a limited period of time. This property returns `true` if the license is a temporary evaluation license, or `false` if the license key is a permanent license.

## Example

The following code sample illustrates using `IsEval` in a script:

```
...
set objCollection cims.LoadLicenses
for each objLics in objCollection
    for each objLic in objLics
        set objKeys = objLic.keys
        i = 0
        do while i < objKeys.Count
            set objKey = objKeys(0)
            'Check for evaluation license keys
            if objKey.isEval then
                wscript.Echo "-- [Eval] ", objKey.ExpiryDate
            end if
            i = i + 1
        loop
    next
    wscript.Echo ""
next
...
```

## IsValid

Determines whether the license key being checked is a valid license.

## Syntax

```
bool IsValid {get;}
```

## Property value

Returns `true` if the license is valid, or `false` if the license key is not valid or has expired.



## Discussion

This property returns true if the license key is a valid license, or false if the license key is invalid. The property also returns false if the license key checked is an expired evaluation license.

## Example

The following code sample illustrates using `IsValid` in a script:

```
...
set objCollection cims.LoadLicenses
for each objLics in objCollection
    for each objLic in objLics
        set objKeys = objLic.keys
        i = 0
        do while i < objKeys.Count
            set objKey = objKeys(0)
            If not objKey.isValid then
                wscript.Echo "Invalid License Key"
                wscript.Quit
            end if
            i = i + 1
        loop
    next
    wscript.Echo ""
next
...
```

## SerialNumber

Gets the serial number of the license key.

## Syntax

```
int SerialNumber {get;}
```

## Property value

The serial number for a specified license key.

## Discussion

The serial number provides a mechanism for tracing which license keys were issued to a recipient.

• • • • •

## Example

The following code sample illustrates using `SerialNumber` in a script:

```
...
set objCollection cims.LoadLicenses
for each objLics in objCollection
    for each objLic in objLics
        set objKeys = objLic.keys
        i = 0
        do while i < objKeys.Count
            set objKey = objKeys(0)
            if objKey.isvalid then
                wscript.Echo "--", "Serial #:",
objKey.SerialNumber
                wscript.Echo "Seats:", objKey.Count
            end if
            i = i + 1
        loop
    next
    wscript.Echo ""
next
...
```

## Type

Gets the license type for a specific license key.

## Syntax

```
LicenseType Type {get;}
```

## Property value

The license type for a license key.

See [Type](#) for possible values.

## Discussion

The license type indicates whether a specific license key is intended for workstation computers or application servers.

## Example

The following code sample illustrates using `Type` in a script:

```
...
set objCollection cims.LoadLicenses
for each objLics in objCollection
```



```
for each objLic in objLics
    set objKeys = objLic.keys
    i = 0
    do while i < objKeys.Count
        set objKey = objKeys(0)
        if objKey.isValid then
            wScript.Echo "License Type", objKey.Type
            wscript.Echo "Serial #:", objKey.SerialNumber
            wScript.Echo "Seats:", objKey.Count
        end if
        i = i + 1
    loop
next
wScript.Echo ""
next
...
```

## Keys

The keys class is used to manage a set of license keys.

## Syntax

```
public interface IKeys
```

## Discussion

This class allows you to retrieve a set of license keys of a particular license type. For example, if you have one or more encrypted license keys (xxxx-xxxx-xxxx) that provide up to 100 licenses of the same license type, such as 100 workstation, server, or application licenses, the keys object can be used to retrieve the key objects that provide those 100 workstation, server, or application licenses.

## Methods

The keys class provides the following methods:

This method	Does this
<code>Add</code>	Adds a license key to the set.
<code>GetEnumerator</code>	Returns an enumeration of <code>Key</code> objects.



This method	Does this
<code>Remove</code>	Removes a license key from the set.

## Properties

The keys class provides the following properties:

This property	Does this
<code>Count</code>	Gets the number of license keys stored in the set.
<code>Item</code>	Gets the license key object using a specific index identifier.

## Add

Adds a license key to the set of keys of a particular type.

## Syntax

```
key Add(string key)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>key</code>	The license key string to add to the set.

## Return value

The added license key object.

## Exceptions

Add may throw one of the following exceptions:

- `ApplicationException` if the license key has expired, is invalid, or is a non-FIPS 140 key on a system that requires FIPS 140 keys.
- `ArgumentException` if the parameter is null or empty or if the key already exists.



## GetEnumerator

Returns an enumeration of key objects.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

The set of key objects.

## Remove

Removes a license key from the set of license keys of a particular type.

### Syntax

```
void Remove(string key)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
key	The license key string to remove from the set.

### Return value

The license key string to be removed.

### Exceptions

Remove may throw one of the following exceptions:

- `ApplicationException` if the license key cannot be found.
- `ArgumentException` if the parameter is null or empty.



## Count

Determines the total number of license keys defined in the collection of keys for a particular type of license.

### Syntax

```
int Count {get;}
```

### Property value

The number of individual license keys included in the keys collection.

## Item

Gets the license key object found at the index point you specify.

### Syntax

```
IKey this[int i] {get;}
```

### Parameter

Specify the following parameter when using this property.

Parameter	Description
i	The index number to use for retrieving the license key object from the set.

### Property value

The license key object.

## License

The `License` class provides access to Centrify license properties.

### Syntax

```
public interface ILicense
```



## Discussion

This class represents the keys of the same license type in the same license container. For example, the `License` object can contain the collection of server, workstation, or application licenses for one license container, such as the default parent container `domain/Program Data/Centrify/Licenses`.

## Properties

The `License` class provides the following properties:

This property	Does this
<code>Count</code>	Determines the total number of licenses of a particular type.
<code>IsEval</code>	Determines whether the license is an evaluation license.
<code>Keys</code>	Gets the license keys associated with the license object.
<code>Type</code>	Gets the license type for the license object.
<code>UsedCount</code>	Gets the total number of licenses that are currently in use.

## Count

Determines the total number of licenses of a particular type.

## Syntax

```
int Count {get;}
```

## Property value

The number of licenses provided in the `License` object.

## Example

The following code sample illustrates using `Count` in a script:

```
...
set objCollection cims.LoadLicenses
for each objLics in objCollection
  for each objLic in objLics
    wscript.Echo "License Type:", objLic.Type
    wscript.Echo "Seats:", objLic.Count
    wscript.Echo "Used:", objLic.UsedCount
  next
wscript.Echo ""
```

• • • • •

next  
...

## IsEval

Determines whether the license is an evaluation license.

### Syntax

```
bool IsEval {get;}
```

### Property value

Returns `true` if the license is a temporary evaluation license, or `false` if it is a permanent license.

### Discussion

An evaluation license provides full use of Centrify software for a limited period of time. This property returns `true` if the license is a temporary evaluation license, or `false` if the license is permanent.

## Keys

Gets the license keys associated with the license object.

### Syntax

```
Keys Keys {get;}
```

### Property value

The keys object that contains the set of individual license keys for this license object.

## Type

Gets the license type for the license object.



## Syntax

```
LicenseType Type {get;}
```

## Property value

The license type.

Possible values:

```
public enum LicenseType
{
    // Not defined
    NotDefined = -1,
    // UNIX workstation license
    workstation = 110,
    // UNIX Server license
    Server = 111,
    // windows Server license
    windowsServer = 112,
    // windows workstation license
    windowsworkstation = 113,
    // Application license for Tomcat
    Tomcat = 210,
    // Application license for JBoss
    JBoss = 211,
    // Application license for WebLogic
    webLogic = 212,
    // Application license for WebSphere
    webSphere = 213,
    // Application license for Apache
    Apache = 214,
    // Application license for DB2
    DB2 = 215,
    // Install evaluation license
    InstallEval = 310,
    // Specific date evaluation license
    SpecificEval = 311
}
```

## Discussion

The license type indicates whether a specific license is intended for workstation computers or application servers.

## Example

The following code sample illustrates using `Type` in a script:

```
...
set objCollection cims.LoadLicenses
for each objLics in objCollection
    for each objLic in objLics
        wscript.Echo "License Type:", objLic.Type
```

• • • • •

```
        wScript.Echo "Seats:", objLic.Count
        wScript.Echo "Used:", objLic.usedCount
    next
    wScript.Echo ""
next
...

```

## UsedCount

Gets the total number of licenses that are currently in use.

### Syntax

```
int UsedCount {get;}
```

### Property value

The total number of licenses that are currently in use.

### Discussion

Available licenses become used licenses when computers join the domain.

### Exceptions

UsedCount throws an `ApplicationException` if license information is unavailable because an LDAP error occurred.

### Example

The following code sample illustrates using UsedCount in a script:

```
...
set objCollection cims.LoadLicenses
for each objLics in objCollection
    for each objLic in objLics
        wScript.Echo "License Type:", objLic.Type
        wScript.Echo "Seats:", objLic.Count
        wScript.Echo "Used:", objLic.usedCount
    next
    wScript.Echo ""
next
...

```



## Licenses

The `Licenses` class is used to manage a set of licenses in a particular license container object.

### Syntax

```
public interface ILicenses
```

### Discussion

This class represents the collection of `License` objects that have been added to this parent container object. The `Licenses` object represents one container in the `LicensesCollection` object.

### Methods

The `Licenses` class provides the following methods:

This method	Does this
<code>AddLicenseKey</code>	Adds a license key to the set of licenses in the license container.
<code>Commit</code>	Commits any changes to the <code>Licenses</code> object to Active Directory.
<code>GetDirectoryEntry</code>	Returns an instance of the directory entry for the <code>Licenses</code> parent container object.
<code>Refresh</code>	Reloads the <code>Licenses</code> object data from the data in Active Directory.
<code>RemoveLicenseKey</code>	Removes a license key from the set.

### Properties

The `Licenses` class provides the following properties:

This property	Does this
<code>Count</code>	Gets the number of license objects stored in this <code>Licenses</code> container.
<code>HasEvaluation</code>	Indicates whether any generated license key installed in the <code>Licenses</code> parent container is an evaluation license.



This property	Does this
<code>HasMachineLicense</code>	Indicates whether any computer license is installed.
<code>ID</code>	Gets the ID from this <code>Licenses</code> object.
<code>IsReadable</code>	Indicates whether the Licenses parent object in Active Directory is readable.
<code>IsWritable</code>	Indicates whether the Licenses parent object in Active Directory is writable.
<code>Item</code>	Gets the <code>License</code> object for a specific type of license.

## AddLicenseKey

Adds a license key to the set of licenses in a particular `Licenses` parent container.

### Syntax

```
IKey AddLicenseKey(string key)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>key</code>	The license key string to add to the parent container.

### Return value

The added key object.

### Exceptions

`AddLicenseKey` may throw one of the following exceptions:

- `ApplicationException` if the license key has expired, is invalid, or is a non-FIPS 140 key on a system that requires FIPS 140 keys.
- `ArgumentException` if the parameter is `null` or empty or if the key already exists.



## Commit

Commits any changes or updates to the `Licenses` object and saves the changes to Active Directory.

### Syntax

```
void Commit()
```

### Discussion

The method does not validate the data before saving it in Active Directory.

### Exceptions

`Commit` throws an `ApplicationException` if the Active Directory domain controller is read-only.

## GetDirectoryEntry

Returns an instance of the `DirectoryEntry` object for the `Licenses` parent container object from Active Directory.

### Syntax

```
DirectoryEntry GetDirectoryEntry ()
```

### Return value

The `DirectoryEntry` object for the `Licenses` parent container object.

### Discussion

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

## Refresh

Reloads the `Licenses` object data from the data in Active Directory.



## Syntax

```
void Refresh()
```

## Discussion

This method refreshes the collection of licenses in the cached object to ensure it is synchronized with the latest information in Active Directory.

## RemoveLicenseKey

Removes a license key from the set of license objects for a particular `Licenses` parent container.

## Syntax

```
void RemoveLicenseKey(string key)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
key	The license key string to remove from the set.

## Return value

The license key object to be removed.

## Exceptions

`RemoveLicenseKey` may throw one of the following exceptions:

- `ApplicationException` if the license key cannot be found.
- `ArgumentException` if the parameter is null or empty.

## Count

Gets the total number of licenses for a particular `Licenses` parent container.

• • • • •

### Syntax

```
int Count {get;}
```

### Property value

The number of licenses provided in the `Licenses` object.

### HasEvaluation

Indicates whether any license key installed in the `Licenses` parent container is an evaluation license.

### Syntax

```
bool HasEvaluation {get;}
```

### Property value

Returns `true` if any generated license key is a temporary evaluation license, or `false` if there are no evaluation license keys installed.

### HasMachineLicense

Indicates whether any computer license is installed.

### Syntax

```
bool HasMachineLicense {get;}
```

### Property value

Returns `true` if any computer license is installed, or `false` if there are no computer licenses installed.

### ID

Gets the ID from the `Licenses` object.



## Syntax

```
string ID {get;}
```

## Property value

The ID.

## IsReadable

Indicates whether the `Licenses` parent object in Active Directory is readable for the current user credentials.

## Syntax

```
bool IsReadable {get;}
```

## Property value

Returns `true` if the `Licenses` parent object is readable, or `false` if the object is not readable.

## Discussion

This property returns a value of `true` if the user accessing the `Licenses` object in Active Directory has sufficient permissions to read its properties.

## IsWritable

Indicates whether the `Licenses` parent object in Active Directory is writable for the current user credentials.

## Syntax

```
bool IsWritable {get;}
```

## Property value

Returns `true` if the `Licenses` parent object is writable, or `false` if the object is not writable.



## Discussion

This property returns a value of `true` if the user accessing the `Licenses` object in Active Directory has sufficient permissions to write its properties.

## Item

Gets the `License` object for a specific type of license.

## Syntax

```
ILicense this[LicenseType type] {get;}
```

## Parameter

Specify the following parameter when using this property.

Parameter	Description
<code>type</code>	The type of <code>License</code> object to retrieve from the <code>Licenses</code> object.

See [Type](#) for possible values.

## Return value

The `License` object of the specified type.

## Discussion

This property enables you to retrieve the `License` object for a collection of licenses of a particular type, such as the collection of server licenses or the collection of licenses for a specific application.

# LicensesCollection

The `LicensesCollection` class is used to manage all of the licenses in all of the `Licenses` parent containers defined for a forest.

## Syntax

```
public interface ILicensesCollection
```



## Discussion

The `LicensesCollection` object is retrieved using the `Cims.LoadLicenses` method.

## Methods

The `LicensesCollection` class provides the following methods:

This method	Does this
<code>Find</code>	Returns the specific parent container object from the collection of all parent license containers in the forest.
<code>GetEnumerator</code>	Returns an enumeration of <code>Licenses</code> objects.
<code>GetLicensedCount</code>	Returns the total number of licenses of the specified license type that have been installed.
<code>GetUsedCount</code>	Returns the total number of licenses of the specified license type that are currently being used.

## Properties

The `LicensesCollection` class provides the following properties:

This property	Does this
<code>Count</code>	Gets the number of license container objects stored in the Active Directory forest.
<code>HasEvaluation</code>	Indicates whether there are any evaluation licenses in the forest.
<code>HasMachineLicense</code>	Indicates whether any computer license is installed.
<code>Item</code>	Gets the parent license container object by index number.

## Find

Returns the specific parent container object from the collection of all parent license containers in the forest.

## Syntax

```
ILicenses Find(DirectoryEntry licenseContainer)
```



## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>LicenseContainer</code>	The <code>Licenses</code> parent container object to retrieve.

## Return value

The specified Licenses container object. The container contains the `$CimsLicenseVersionX` object.

## Discussion

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

## GetEnumerator

Returns an enumeration of `Licenses` objects.

## Syntax

```
IEnumerator GetEnumerator()
```

## Return value

The set of `Licenses` objects.

## GetLicensedCount

Returns the total number of licenses of the specified license type that have been installed in an Active Directory forest.

## Syntax

```
int GetLicensedCount(LicenseType type)
```

## Parameter

Specify the following parameter when using this method:



---

Parameter	Description
type	The license type for which you want to get a complete count.

---

See [Type](#) for possible values.

## Return value

Returns a value that represents the number of licenses installed of the specified type.

## GetUsedCount

Returns the total number of licenses of the specified license type that are currently being used in an Active Directory forest.

## Syntax

```
int GetUsedCount(LicenseType type)
```

## Parameter

Specify the following parameter when using this method:

---

Parameter	Description
type	The license type for which you want to determine the number being used.

---

See [Type](#) for possible values.

## Return value

Returns a value that represents the number of licenses of the specified type that are currently in use.

## Exceptions

`GetUsedCount` throws an `ApplicationException` if license information is unavailable because an LDAP error occurred.

## Count

Gets the total number of parent license containers in the Active Directory forest.

• • • • •

### Syntax

```
int Count {get;}
```

### Property value

The number of parent license containers in the collection of parent containers.

### HasEvaluation

Indicates whether any generated license key installed in any of the parent license containers is an evaluation license.

### Syntax

```
bool HasEvaluation {get;}
```

### Property value

Returns true if any evaluation license is installed, or false if there are no evaluation licenses installed.

### HasMachineLicense

Indicates whether any computer license is installed.

### Syntax

```
bool HasMachineLicense {get;}
```

### Property value

Returns true if any computer license is installed, or false if there are no computer licenses installed.

### Item

Gets the parent license container object by index number.



## Syntax

```
ILicenses this[int index] {get;}
```

## Parameter

Specify the following parameter when using this property.

Parameter	Description
index	The index number for retrieving the parent license container object

## Return value

The parent container object found at the specified index number.

# MzRoleAssignment

Represents a computer-level role assignment.

## Syntax

```
public interface IMzRoleAssignment : IRoleAssignment
```

## Methods

The MzRoleAssignment class provides the following methods:

This method	Does this
<code>ClearCustomAttributes</code>	VBScript interface to clear the custom attributes for this class. (Inherited from <a href="#">RoleAssignment</a> .)
<code>Commit</code>	Commits changes in the role assignment to Active Directory. (Inherited from <a href="#">RoleAssignment</a> .)
<code>Delete</code>	Deletes the role. (Inherited from <a href="#">RoleAssignment</a> .)
<code>GetComputer</code>	Returns the computer for which the role assignment is made.



This method	Does this
<code>GetTrustee</code>	Returns the trustee being assigned. (Inherited from <code>RoleAssignment</code> .)
<code>ICustomAttributeContainer</code> <code>GetCustomAttributeContainer</code>	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class. (Inherited from <code>RoleAssignment</code> .)
<code>SetCustomAttribute</code>	VBScript interface to set the custom attributes for this class. (Inherited from <code>RoleAssignment</code> .)
<code>Validate</code>	Validates the role assignment. (Inherited from <code>RoleAssignment</code> .)

## Properties

The `MzRoleAssignment` class provides the following properties:

This property	Does this
<code>CustomAttributes</code>	VBScript only: Gets or sets custom attributes for this class.
<code>EndTime</code>	Determines the time at which this role becomes inactive. (Inherited from <code>RoleAssignment</code> .)
<code>Id</code>	Gets the GUID of the role assignment. (Inherited from <code>RoleAssignment</code> .)
<code>IsRoleOrphaned</code>	Indicates whether the role assignment is orphaned due to a missing or invalid role. (Inherited from <code>RoleAssignment</code> .)
<code>IsTrusteeOrphaned</code>	Indicates whether the role assignment is orphaned due to a missing or invalid trustee. (Inherited from <code>RoleAssignment</code> .)
<code>LocalTrustee</code>	Gets or sets the local trustee being assigned. (Inherited from <code>RoleAssignment</code> .)
<code>Role</code>	Gets or sets the role the trustee is assigned to. (Inherited from <code>RoleAssignment</code> .)
<code>StartTime</code>	Gets or sets the time from which this role becomes effective. (Inherited from <code>RoleAssignment</code> .)
<code>TrusteeDn</code>	Gets or sets the distinguished name of the trustee assigned this role.



This property	Does this
	(Inherited from <a href="#">RoleAssignment</a> .)
<a href="#">TrusteeType</a>	Gets or sets the trustee type of the role assignment. (Inherited from <a href="#">RoleAssignment</a> .)

## GetComputer

Returns the computer for which the role assignment is made.

### Syntax

```
IComputer GetComputer()
```

### Return value

The computer object representing the computer for which the role assignment is made.

### Exceptions

`GetComputer` throws an `ApplicationException` if there are multiple computers, computer service connection points (SCPs), or zones that have the same DNS host name; if the method failed to find the computer; or if the method failed to get the computer profile from the role assignment.

## NetworkAccess

This class represents a network access right.

### Syntax

```
public interface INetworkAccess:IRight
```

The `NetworkAccess` class provides the following methods:



This method	Does this
<code>Commit</code>	Commits changes in the right to Active Directory. (Inherited from <code>Right</code> .)
<code>Delete</code>	Removes the right. (Inherited from <code>Right</code> .)

## Properties

The `NetworkAccess` class provides the following properties:

This property	Does this
<code>Description</code>	Gets or sets the description of the right. (Inherited from <code>Right</code> .)
<code>IsReadable</code>	Indicates whether the right is readable. (Inherited from <code>Right</code> .)
<code>IsWritable</code>	Indicates whether the right is writable. (Inherited from <code>Right</code> .)
<code>Name</code>	Gets or sets the name of the right. (Inherited from <code>Right</code> .)
<code>Priority</code>	Gets or sets the priority of this right.
<code>RequirePassword</code>	Gets or sets whether the user's password is required when this right is used.
<code>RunAs</code>	Gets or sets the SID for the run-as user or an SID for the user assigned the right (VBScript).
<code>RunAsList</code>	Gets or sets the SID for the run-as user or a list of SIDs for the users assigned the right (.NET).
<code>RunAsType</code>	Gets or sets the run-as type for the right.
<code>Zone</code>	Gets the zone this right belongs to. (Inherited from <code>Right</code> .)

## Discussion

A network access right enables a user to run an application on a remote computer as another user. For example, a network access right can give a user the ability to run as an SQL Administrator on a remote server.



## Priority

Gets or sets the priority of this right.

### Syntax

```
int Priority {get; set;}
```

### Property value

The priority of the right. Default is 0.

### Discussion

This number is used when handling multiple matches for rights specified by wild cards. If rights specified by this property object match rights specified by another property object, the object with the higher priority prevails. The higher the value of the `Priority` property, the higher the priority.

## RequirePassword

Gets or sets whether the logged-in user's password is required when this right is used.

### Syntax

```
bool RequirePassword {get; set;}
```

### Property value

Set to `true` if the right requires the logged-in user's password.

## RunAs

Gets or sets the run-as property for this right.

### Syntax

```
string RunAs {get; set;}
```



## Property value

The run-as property for a single user.

## Discussion

If the `RunAsType` property is set to `self`, the remote application is run under the logged-in user account, but with the additional privileges of the user whose SID is listed in the `RunAs` property. For example, if the `NetworkAccess` right is set to run as `self` and `RunAs` contains the SID of the Network Admins group, then this application runs with the permissions of the logged-in user plus the permissions of the Network Admins group.

If the `RunAsType` property is set to `User`, the remote application is run under the user whose SID is listed in the `RunAs` property. For example, if the `NetworkAccess` right is set to run as `User` and `RunAs` contains the SID of the user `NetAdmin`, then this application runs with the permissions of the `NetAdmin` user.

If the `RunAs` property is empty, this right is invalid and an exception is thrown when you call the `Commit` method.

**Note:** This property is for use in VBScript programs. Use the `RunAsList` property for .NET.

## RunAsList

Gets or sets the run-as list for this right.

## Syntax

```
IList<SecurityIdentifier> RunAsList {get; set;}
```

## Property value

The run-as list for the right.

## Discussion

If the `RunAsType` property is set to `self`, the remote application is run under the logged-in user account, but with the additional privileges of the groups whose SIDs are listed in the `RunAsList` property. For example, if the `NetworkAccess` right is set to run as `self` and `RunAsList` contains the SID of



the Network Admins group, then this application runs with the permissions of the logged-in user plus the permissions of the Network Admins group.

If the `RunAsType` property is set to `User`, the remote application is run under the user whose SID is listed in the `RunAsList` property. In this case, the `RunAsList` property contains only a single SID. For example, if the `NetworkAccess` right is set to run as `User` and `RunAsList` contains the SID of the user `NetAdmin`, then this application runs with the permissions of the `NetAdmin` user.

If the `RunAsList` property is empty, this right is invalid and an exception is thrown when you call the `Commit` method.

**Note:** This property can only be used in .NET programs. Use the `RunAs` property for VBScript.

## RunAsType

Gets or sets the run-as type for this right.

### Syntax

```
WindowsRunAsType RunAsType {get; set;}
```

### Property value

The run-as type of the right.

Possible values:

```
public enum WindowsRunAsType
{
    // Run as self
    Self,
    // Run as another user
    User
}
```

### Discussion

If the `RunAsType` property is set to `Self`, the remote application runs as the logged-in user with the additional privileges of the groups whose SIDs are listed in the `RunAsList` property. For example, if the `NetworkAccess` right is set to run as `Self` and `RunAsList` contains the SID of the Network Admins group, then this application runs with the permissions of the logged-in user plus the permissions of the Network Admins group.



If the `RunAsType` property is set to `User`, the application is run as the user whose SID is listed in the `RunAsList` property. For example, if the `NetworkAccess` right is set to run as `User` and `RunAsList` contains the SID of the user `NetAdmin`, then this application runs as `NetAdmin` with the permissions of that user.

## NetworkAccesses

The `NetworkAccesses` class manages a collection of network access rights.

### Syntax

```
public interface INetworkAccesses
```

### Methods

The `NetworkAccesses` class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Gets the enumerator you can use to enumerate all network access rights.

### GetEnumerator

Returns an enumeration of `NetworkAccess` objects.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

Returns an enumerator you can use to list all the `NetworkAccess` objects.

## Pam

Represents a PAM application access right.



## Syntax

```
public interface IPam: IRight
```

## Discussion

A PAM (Pluggable Authentication Module) application right gives a user the ability to access the authorized PAM-enabled application.

## Methods

The Pam class provides the following methods:

This method	Does this
<code>Commit</code>	Commits changes in the right to Active Directory. (Inherited from <code>Right</code> .)
<code>Delete</code>	Deletes the right. (Inherited from <code>Right</code> .)

## Properties

The Pam class provides the following properties:

This property	Does this
<code>Application</code>	Gets or sets the PAM application.
<code>Description</code>	Gets or sets the description of the right. (Inherited from <code>Right</code> .)
<code>IsReadable</code>	Indicates whether the right is readable. (Inherited from <code>Right</code> .)
<code>IsWritable</code>	Indicates whether the right is writable. (Inherited from <code>Right</code> .)
<code>Name</code>	Gets or sets the name of the right. (Inherited from <code>Right</code> .)
<code>Zone</code>	Gets the zone this right belongs to. (Inherited from <code>Right</code> .)



## Application

Gets or sets the PAM application for which this is a right.

### Syntax

```
string Application {get; set;}
```

### Property value

The file path of the application.

### Exceptions

`Application` throws an `ArgumentException` if you try to set the property and the string is null or empty.

### Example

The following code sample illustrates using `Application` in a script:

```
...
string strParent = "CN=zones,CN=Centrify,CN=Program Data";
if (args.Length != 3)
{
    Console.WriteLine("Usage:");
    Console.WriteLine(" test_add_pam.exe \"zone-name\" \"pam-name\" \"pam-application\"");
    return;
}
string strZone = args[0];
string strName = args[1];
string strApp = args[2];
// Need to obtain an active directory container object
DirectoryEntry objRootDSE = new DirectoryEntry("LDAP://rootDSE");
DirectoryEntry objContainer = new DirectoryEntry("LDAP://" + strParent +
", " +
objRootDSE.Properties["defaultNamingContext"].Value.ToString());
string strContainerDN = objContainer.Properties
["DistinguishedName"].Value as string;
// Create a CIMS object to interact with AD
ICims cims = new Cims();
// Get the zone object
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + ", " + strContainerDN) as
IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
}
else
{
    IPam objPam = objZone.GetPamAccess(strName);
}
```



```
if (objPam != null)
{
    Console.WriteLine("PAM " + strName + " already exist.");
}
else
{
    objPam = objZone.CreatePamAccess();
    objPam.Name = strName;
    objPam.Application = strApp;
    objPam.Description = "optional description";
    objPam.Commit();
}
}
...
```

## Pams

The Pams class manages a collection of PAM application access rights.

### Syntax

```
public interface IPams
```

### Methods

The Pams class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Returns the enumerator you can use to enumerate all PAM rights.

### GetEnumerator

Returns an enumeration of PAM access rights.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

An enumerator you can use to list all the Pam objects.



## Right

This class provides a base class for all rights.

### Syntax

```
public interface IRight
```

### Methods

The Right class provides the following methods:

This method	Does this
<code>Commit</code>	Commits changes in the right to Active Directory.
<code>Delete</code>	Deletes the right.

### Properties

The Right class provides the following properties:

This property	Does this
<code>Description</code>	Gets or sets the description of the right.
<code>IsReadable</code>	Indicates whether the right is readable.
<code>IsWritable</code>	Indicates whether the right is writable.
<code>Name</code>	Gets or sets the name of the right.
<code>Zone</code>	Gets the zone this right belongs to.

### Commit

Commits any changes or updates to the `Right` object and saves the changes to Active Directory.

### Syntax

```
void Commit()
```



## Discussion

The method does not validate the data before saving it in Active Directory.

## Exceptions

`Commit` throws an `ApplicationException` if:

- the command right name or command pattern is null, empty, or invalid
- the command name is null or empty
- the command path is null or empty
- the method cannot find the command right or authorization data for the zone

If the `Commit` method fails, see the message returned by the exception for more specific information about the reason for the failure.

## Delete

Deletes the right from Active Directory.

## Syntax

```
void Delete()
```

## Exceptions

`Delete` throws an `ApplicationException` if the method cannot find the command right to delete or cannot find authorization data for the zone.

If the `Delete` method fails, see the message returned by the exception for more specific information about the reason for the failure.

## Description

Gets or sets the description of the right.

## Syntax

```
string Description {get; set;}
```

• • • • •

### Property value

A string describing the right.

### IsReadable

Indicates whether the right is readable.

### Syntax

```
bool IsReadable {get;}
```

### Property value

Returns true if the right is readable.

### IsWritable

Indicates whether the right is writable.

### Syntax

```
bool IsWritable {get;}
```

### Property value

Returns true if the right is writable.

### Name

Gets or sets the name of the right.

### Syntax

```
string Name {get; set;}
```

### Property value

The name of the right. The name can contain only letters (upper- or lowercase), numerals 0 through 9, and the hyphen (-) and underscore (\_) characters.



## Exceptions

Name throws an `ArgumentException` if the name is null, empty, or contains invalid characters.

## Zone

Indicates the zone to which this right belongs.

## Syntax

```
IZone Zone {get;}
```

## Property value

The zone.

## Role

The `Role` class represents a user access role.

## Syntax

```
public interface IRole
```

## Methods

The `Role` class provides the following methods:

This method	Does this
<code>AddCommand</code>	Adds a command right to the role.
<code>AddNetworkAccess</code>	Adds a network application access right to the role.
<code>AddPamAccess</code>	Adds a PAM application access right to the role.
<code>AddSsh</code>	Adds an SSH application access right to the role.
<code>AddwindowsApplication</code>	Adds a Windows application right to the role.
<code>AddwindowsDesktop</code>	Adds a Windows desktop right to the role.
<code>Assign</code>	Adds a trustee to the role at the zone or computer level.



This method	Does this
<code>ClearCustomAttributes</code>	VBScript interface to clear the custom attributes for this class.
<code>Commit</code>	Commits changes to the role to Active Directory.
<code>Delete</code>	Deletes the role from Active Directory.
<code>GetCommands</code>	Returns all command rights added to the role.
<code>ICustomAttributeContainer</code> <code>GetCustomAttributeContainer</code>	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class.
<code>GetNetworkAccesses</code>	Returns the collection of all network application access rights added to this role.
<code>GetPamAccesses</code>	Returns all PAM application access rights added to the role.
<code>GetSshRights</code>	Returns all SSH application access rights added to the role.
<code>GetWindowsApplications</code>	Returns the collection of all Windows application rights added to this role.
<code>GetWindowsDesktops</code>	Returns the collection of all Windows desktop rights added to this role.
<code>IsApplicable</code>	Indicates whether the role is valid in a specified time period.
<code>RemoveAllRights</code>	Removes all rights from the role.
<code>RemoveCommand</code>	Removes a specific command right from the role.
<code>RemoveNetworkAccess</code>	Removes a specific PAM application right from the role.
<code>RemovePamAccess</code>	Removes a specific PAM application right from the role.
<code>RemoveSshRight</code>	Removes a specific SSH application right from the role.
<code>RemoveWindowsApplication</code>	Removes a specific Windows application access right from the role.
<code>RemoveWindowsDesktop</code>	Removes a specific Windows desktop right from the role.
<code>GetSshRight</code>	Sets a day of the week on which the role is active or inactive.
<code>SetApplicableHour</code>	Sets a day and hour of the week for which the role is active or inactive.
<code>SetCustomAttribute</code>	VBScript interface to set the custom attributes for this class.

## Properties

The `Role` class provides the following properties:



This property	Does this
<code>AllowLocalUser</code>	Determines whether the role allows local users.
<code>ApplicableTimeHexString</code>	Gets or sets the time at which the role is active, in hex format.
<code>CustomAttributes</code>	VBScript only: Gets or sets custom attributes for this class.
<code>Description</code>	Gets or sets the description of the role.
<code>Guid</code>	Gets the GUID for this role.
<code>IsReadable</code>	Indicates whether the role is readable.
<code>IsWritable</code>	Indicates whether the role is writable.
<code>Name</code>	Gets or sets the name of the role.
<code>SystemRights</code>	Gets or sets the system rights granted to the role.
<code>Zone</code>	Gets the zone to which this role belongs.

## AddCommand

Adds a command right to the role.

### Syntax

```
void AddCommand(Icommand command)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>command</code>	The command right you want to add to the role.

### Discussion

This command right is not stored in Active Directory until you call the `Commit` method.

### Exceptions

`AddCommand` throws an `ApplicationException` if the command right is not in the current or parent zone.

### Example

The following code sample illustrates using `AddCommand` in a script:

• • • • •

```
...
// Get the zone object
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
    return;
}
IRole objRle = objZone.GetRole(strRole);
    if (objRole == null)
{
    Console.WriteLine("Role " + strRole + " does not exist.");
    return;
}
ICommand objCmd = objZone.GetCommand(strCmd);
if (objCmd == null)
{
    Console.WriteLine("Command " + strCmd + " does not exist.");
    return;
}
objRole.AddCommand(objCmd);
objRole.Commit();
...
```

## AddNetworkAccess

Adds a network application access right to the role.

### Syntax

```
void AddNetworkAccess(INetworkAccess networkAccess)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
networkAccess	The network application right you want to add to the role.

### Discussion

This right is not stored in Active Directory until you call the `Commit` method.



## Exceptions

AddNetworkAccess throws an `ApplicationException` if the network access right is not in the current or parent zone.

## AddPamAccess

Adds a PAM application access right to the role.

### Syntax

```
void AddPamAccess(IPam pam)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
pam	The PAM application right you want to add to the role.

### Discussion

This right is not stored in Active Directory until you call the `Commit` method.

## Exceptions

AddPamAccess throws an `ApplicationException` if the PAM application access right is not in the current or parent zone.

## AddSsh

Adds an SSH application access right to the role.

### Syntax

```
void AddSsh(ISsh ssh)
```

### Parameter

Specify the following parameter when using this method:



---

Parameter	Description
ssh	The SSH application right you want to add to the role.

---

## Discussion

This right is not stored in Active Directory until you call the `Commit` method.

## Exceptions

`AddSsh` throws an `ApplicationException` if the SSH application right is not in the current or parent zone.

## AddWindowsApplication

Adds a Windows application right to the role.

## Syntax

```
void AddWindowsApplication(IWindowsApplication windowsApplication)
```

## Parameter

Specify the following parameter when using this method:

---

Parameter	Description
windowsApplication	The Windows application right you want to add to the role.

---

## Discussion

This right is not stored in Active Directory until you call the `Commit` method.

## Exceptions

`AddWindowsApplication` throws an `ApplicationException` if the Windows application right is not in the current or parent zone.

## AddWindowsDesktop

Adds a Windows desktop right to the role.



## Syntax

```
void AddWindowsDesktop(IWindowsDesktop windowsDesktop)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
windowsDesktop	The Windows desktop right you want to add to the role.

## Discussion

This right is not stored in Active Directory until you call the `Commit` method.

## Exceptions

`AddWindowsDesktop` throws an `ApplicationException` if the Windows desktop right is not in the current or parent zone.

## Example

The following code sample illustrates using `AddWindowsDesktop` in a script:

```
...
// Get the zone object
IHierarchicalZone objZone =
    cims.GetZoneByPath("cn=" + strZone + "," + strContainerDN) as
IHierarchicalZone;
if (objZone == null)
{
    Console.WriteLine("Zone " + strZone + " does not exist.");
    return;
}
IRole objRole = objZone.GetRole(strRole);
    if (objRole == null)
{
    Console.WriteLine("Role " + strRole + " does not exist.");
    return;
}
IWindowsDesktop objWindowsDesktop = objZone.GetWindowsDesktop
(strWindowsDesktop);
if (objWindowsDesktop == null)
{
    Console.WriteLine("WindowsDesktop " + strWindowsDesktop + "
does not exist.");
    return;
}
objRole.AddWindowsDesktop(objWindowsDesktop);
objRole.Commit();
...
```



## Assign

Assigns a trustee to a role at the zone or computer level.

### Syntax

```
IRoleAssignment Assign(DirectoryEntry trusteeDE, IComputer computer)
```

```
IRoleAssignment Assign(DirectoryEntry trusteeDE, IZone zone)
```

```
IRoleAssignment Assign(SearchResult trusteeSR, IComputer computer)
```

```
IRoleAssignment Assign(SearchResult trusteeSR, IZone zone)
```

```
IRoleAssignment Assign(string trusteeDN, IComputer computer)
```

```
IRoleAssignment Assign(string trusteeDN, IZone zone)
```

### Parameters

Use the following parameters with this method.

Parameter	Description
trusteeDE	The directory entry for the trustee (user or group) you want to add.
trusteeSR	The directory entry for a trustee specified as a search result.
trusteeDn	The trustee specified as a distinguished name.
computer	The computer to which you want to add the role.
zone	The zone to which you want to add the role.

### Return value

The role assignment that includes the specified trustee. This role assignment is not stored in Active Directory until you call the `RoleAssignment.Commit` method.

### Discussion

The `Assign(DirectoryEntry trusteeDE, IComputer computer)`, `Assign(SearchResult trusteeSr, IComputer computer)`, `Assign(DirectoryEntry trusteeDE, IZone zone)` and `Assign(SearchResult trusteeSr, IZone zone)` methods are available only for .NET-based programs.



## Exceptions

Assign may throw one of the following exceptions:

- `ApplicationException` if the trustee is not a user or a group, you attempt to assign a role to a zone other than the containing or child zone, the method fails to create a role assignment (see the message returned by the exception for the reason), the method cannot find the trustee object or distinguished name in the specified search result, or the method cannot find the trustee.
- `ArgumentException` if any parameter is null or empty.

## Commit

Commits any changes or updates to the `Role` object and saves the changes to Active Directory.

## Syntax

```
void Commit()
```

## Discussion

The method does not validate the data before saving it in Active Directory.

## Exceptions

`Commit` throws an `ApplicationException` if:

- the role name is null, empty, or invalid
- the method cannot find the role or command right
- the method cannot find authorization data for the zone

If the `Commit` method fails, see the message returned by the exception for more specific information about the reason for the failure.

## Delete

Deletes the role from Active Directory.



## Syntax

```
void Delete()
```

## Exceptions

Delete throws an `ApplicationException` if the method cannot find the role or authorization data for the zone.

If the Delete method fails, see the message returned by the exception for more specific information about the reason for the failure.

## GetCommands

Returns all the command rights added to this role.

## Syntax

```
ICollection GetCommands()
```

## Return value

The collection of command rights. Enumerate this object to get all of the `ICommand` objects for this role.

## GetNetworkAccesses

Returns the collection of all network application access rights added to this role.

## Syntax

```
ICollection GetNetworkAccesses()
```

## Return value

A collection of `NetworkAccess` objects representing all the network application access rights in this role.



## GetPamAccesses

Returns the collection of all PAM application rights added to this role.

### Syntax

```
IPams GetPamAccesses()
```

### Return value

A collection of **Pam** objects representing all the PAM application rights in this role.

## GetSshRights

Returns the collection of all SSH application rights added to this role.

### Syntax

```
ISshs GetSshRights()
```

### Return value

A collection of **Ssh** objects representing all the SSH application rights in this role.

## GetWindowsApplications

Returns the collection of all Windows application rights added to this role.

### Syntax

```
IWindowsApplications GetWindowsApplications()
```

### Return value

A collection of **WindowsApplication** objects representing all the Windows application rights in this role.



## GetWindowsDesktops

Returns the collection of all Windows desktop rights added to this role.

### Syntax

```
IWindowsDesktops GetWindowsDesktops()
```

### Return value

A collection of WindowsDesktop objects representing all the Windows desktop rights in this role.

## IsApplicable

Checks to see if the role is valid at a specified time.

### Syntax

```
bool IsApplicable(int dayOfWeek, int hourOfDay)
```

### Parameter

Specify the following parameters when using this method.

Parameter	Description
dayOfWeek	The day of the week, where 0 is Sunday and 6 is Saturday.
hourOfDay	The hour of the day, where 0 is midnight and 23 is 11:00 PM

### Return value

Returns true if the role is valid at the specified time.

### Discussion

If the role is active at a particular hour, it is active for that entire hour. To set a specific hour, see [SetApplicableHour](#). To set an entire day, see [GetSshRight](#). To set up an entire schedule with one call, see [ApplicableTimeHexString](#).



## RemoveAllRights

Removes all rights from the role.

### Syntax

```
void RemoveAllRights()
```

### Discussion

Changes to the role assignments are not stored in Active Directory until you call the `Commit` method.

## RemoveCommand

Removes a specific command right from the role.

### Syntax

```
void RemoveCommand(ICommand command)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>command</code>	The command right you want to remove.

### Discussion

Changes to the role assignments are not stored in Active Directory until you call the `Commit` method.

## RemoveNetworkAccess

Removes a specific network access right from the role.

### Syntax

```
void RemoveNetworkAccess(INetworkAccess networkAccess)
```



## Parameter

Specify the following parameter when using this method:

Parameter	Description
networkAccess	The network application access right you want to remove.

## Discussion

Changes to the role assignments are not stored in Active Directory until you call the `Commit` method.

## RemovePamAccess

Removes a specific PAM application access right from the role.

## Syntax

```
void RemovePamAccess(IPam pam)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
pam	The PAM application access right you want to remove.

## Discussion

Changes to the role assignments are not stored in Active Directory until you call the `Commit` method.

## RemoveSshRight

Removes a specific SSH application access right from the role.

## Syntax

```
void RemoveSshRight(ISsh ssh)
```



## Parameter

Specify the following parameter when using this method:

Parameter	Description
ssh	The SSH application access right you want to remove.

## Discussion

Changes to the role assignments are not stored in Active Directory until you call the `Commit` method.

## RemoveWindowsApplication

Removes a specific Windows application access right from the role.

## Syntax

```
void RemoveWindowsApplication(IWindowsApplication windowsApplication)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
windowsApplication	The Windows application access right you want to remove.

## Discussion

Changes to the role assignments are not stored in Active Directory until you call the `Commit` method.

## RemoveWindowsDesktop

Removes a specific Windows desktop right from the role.

## Syntax

```
void RemoveWindowsDesktop(IWindowsDesktop windowsDesktop)
```



## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>windowsDesktop</code>	The Windows desktop right you want to remove.

## Discussion

Changes to the role assignments are not stored in Active Directory until you call the `Commit` method.

## SetApplicableDay

Sets a day of the week on which the role is active or inactive.

## Syntax

```
void setApplicableDay(int dayOfWeek, bool isApplicable)
```

## Parameter

Specify the following parameters when using this method.

Parameter	Description
<code>dayOfWeek</code>	The day of the week on which you want the role to be active or inactive, where 0 is Sunday and 6 is Saturday.
<code>isApplicable</code>	Set <code>true</code> to make the role active on the specified day, or <code>false</code> to make the role inactive on that day.

## Discussion

If you set the role active on Monday, it is active all day each Monday. To set a specific hour, see [SetApplicableHour](#). To set up an entire schedule with one call, see [ApplicableTimeHexString](#). To check whether the role is active at a given time, see [IsApplicable](#).

Changes to the role assignments are not stored in Active Directory until you call the `Commit` method.



## SetApplicableHour

Sets a day and hour of the week for which the role is active or inactive.

### Syntax

```
void SetApplicableHour(int dayOfWeek, int hourOfDay, bool  
isApplicable)
```

### Parameter

Specify the following parameters when using this method.

Parameter	Description
dayOfWeek	The day of the week on which you want the role to be active or inactive, where 0 is Sunday and 6 is Saturday.
hourOfDay	The hour of the day at which you want the role to be active or inactive, where 0 is midnight and 23 is 11:00 PM
isApplicable	Set <code>true</code> to make the role active on the specified day and hour, or <code>false</code> to make the role inactive.

### Discussion

If you set the role active on Monday at 10 AM, it is active every Monday from 10:00 to 10:59. To set an entire day, see [GetSshRight](#). To set up an entire schedule with one call, see [ApplicableTimeHexString](#). To check whether the role is active at a given time, see [IsApplicable](#).

Changes to the role assignments are not stored in Active Directory until you call the `Commit` method.

## AllowLocalUser

Determines whether the role allows local users.

### Syntax

```
bool AllowLocalUser {get; set;}
```

### Property value

Set to `true` if the role allows local users. The default is `false`.



## ApplicableTimeHexString

Gets or sets the time at which the role is active, specified as a hexadecimal number.

### Syntax

```
string ApplicableTimeHexString {get; set;}
```

### Property value

The times at which the role is active or inactive.

### Discussion

This is a 42-character (21-byte) hexadecimal value stored as a string. When the hex value is converted to a binary value, its 168 bits each map to a single hour within the week. If a bit is set to 1, its corresponding hour is enabled for the role. If set to 0, its corresponding hour is disabled.

For details of how the bits are mapped to the hours of the week, see [Reading and setting timebox values](#)

To set a specific hour, see [SetApplicableHour](#). To set an entire day, see [GetSshRight](#). To check whether the role is active at a given time, see [IsApplicable](#).

### Exceptions

`ApplicableTimeHexString` throws an `ArgumentException` if the hex string is invalid.

## Description

Gets or sets a description of the role.

### Syntax

```
string Description {get; set;}
```

### Property value

A description of the role.

• • • • •

## Guid

Gets the GUID for this role.

### Syntax

```
Guid Guid {get;}
```

### Property value

The GUID. If the role has not been saved, this property returns `Guid.Empty`.

## IsReadable

Indicates whether the role is readable.

### Syntax

```
bool IsReadable {get;}
```

### Property value

Returns `true` if the role is readable.

## Discussion

This property returns a value of `true` if the user accessing the `Role` object in Active Directory has sufficient permissions to read its properties.

## IsWritable

Indicates whether the role is writable.

### Syntax

```
bool IsWritable {get;}
```

### Property value

Returns `true` if the role is writable.



## Discussion

This property returns a value of `true` if the user accessing the `Role` object in Active Directory has sufficient permissions to write its properties.

## Name

Gets or sets the name of the role.

## Syntax

```
string Name {get; set;}
```

## Property value

The name of the role. The name can contain only letters (upper- or lowercase), numerals 0 through 9, and the hyphen (-) and underscore (\_) characters.

## Exceptions

`Name` throws an `ArgumentException` if the name is `null` or empty or contains invalid characters.

## SystemRights

Gets or sets system rights granted to the role.

## Syntax

```
SystemRight SystemRights {get; set;}
```

## Property value

A byte indicating which system rights are granted.

Possible values:

```
public enum SystemRight
{
    // No system rights
    None = 0,
    // Log in with password
    LoginWithPassword = 1,
    // Log in without password (single sign-on)
    LoginWithoutPassword = 2,
```



```
// Ignore disabled status in Active Directory and log in
anyway
    IgnoreDisabled = 4,
// Allow using a full shell
    AllowNonRestrictedShell = 8,
// NoAudit
    NoAudit = 16,
// Audit always required
    AuditRequired = 32
// Multi-factor authentication required
    MfaRequired = 512,
// Permit login when running in emergency mode
    Rescue = 64
// Allow logging in from the console
    ConsoleLogon = 128
// Allow logging in remotely (RDP)
    RemoteLogon = 256
// Allow powershell remote access
    PsRemote = 1024
}
```

## Discussion

The Rescue system right allows the user to log in when there are problems with the authorization cache or the auditing service that are preventing all other users from logging in. For example, if auditing is required but the auditing service is not running or not available, only users with the rescue system right will be allowed to log in. The rescue system right requires the Centrify NSS module to be running in “emergency” mode because the `adcli` process is not running.

## Zone

Gets the zone to which this role belongs.

## Syntax

```
IZone Zone {get;}
```

## Property value

The zone.

## RoleAssignment

This class represents a zone-level role assignment.



## Syntax

```
public interface IRoleAssignment
```

## Methods

The RoleAssignment class provides the following methods:

This method	Does this
<code>ClearCustomAttributes</code>	VBScript interface to clear the custom attributes for this class.
<code>Commit</code>	Commits changes in the role assignment to Active Directory.
<code>Delete</code>	Deletes the role assignment.
<code>ICustomAttributeContainer</code> <code>GetCustomAttributeContainer</code>	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class.
<code>GetTrustee</code>	Returns the trustee being assigned.
<code>SetCustomAttribute</code>	VBScript interface to set the custom attributes for this class.
<code>validate</code>	Validates the role assignment

## Properties

The RoleAssignment class provides the following properties:

This property	Does this
<code>CustomAttributes</code>	VBScript only: Gets or sets custom attributes for this class.
<code>EndTime</code>	Determines the time at which this role assignment becomes inactive.
<code>Id</code>	Gets the GUID of the role assignment.
<code>IsRoleOrphaned</code>	Indicates whether the role assignment is orphaned due to a missing or invalid role.
<code>IsTrusteeOrphaned</code>	Indicates whether the role assignment is orphaned due to a missing or invalid trustee.
<code>LocalTrustee</code>	Gets or sets the local trustee being assigned.
<code>Role</code>	Gets or sets the role the trustee is assigned to.
<code>StartTime</code>	Gets or sets the time from which this role assignment becomes effective.



This property	Does this
<code>TrusteeDn</code>	Gets or sets the distinguished name of the trustee assigned this role.
<code>TrusteeType</code>	Gets or sets the trustee type of the role assignment.

## Commit

Commits any changes or updates to the role assignment and saves the changes to Active Directory.

### Syntax

```
void Commit()
```

### Discussion

The method does not validate the data before saving it in Active Directory. Call the `Validate` method before calling the `Commit` method to make sure the data is valid.

### Exceptions

`Commit` throws an `ApplicationException` if:

- the role assignment already exists
- the method cannot find the role assignment or the role
- the method cannot find authorization data for the zone

If the `Commit` method fails, see the message returned by the exception for more specific information about the reason for the failure.

## Delete

Deletes the role assignment from Active Directory.

### Syntax

```
void Delete()
```



## Exceptions

`Delete` throws an `ApplicationException` if the method cannot find the role assignment or authorization data for the zone.

If the `Delete` method fails, see the message returned by the exception for more specific information about the reason for the failure.

## GetTrustee

Returns the trustee assigned to the role.

### Syntax

```
DirectoryEntry GetTrustee()
```

### Return value

The directory entree of the user or group assigned by this role assignment.

## Exceptions

`GetTrustee` throws an `ApplicationException` if the method fails to get the trustee object from directory services.

## Validate

Validates the data in the role assignment object before any changes are committed to Active Directory.

### Syntax

```
void validate()
```

## Exceptions

`validate` throws an `ApplicationException` if:

- the role assignment already exists
- the role is `null`
- the method cannot find the role assignment or the role



- the method cannot find authorization data for the zone,
- the start time is later than the end time

If the `validate` method fails, see the message returned by the exception for more specific information about the reason for the failure.

## EndTime

Gets or sets the time after which this role assignment is inactive.

### Syntax

```
DateTIme EndTIme {get; set;}
```

### Property value

The time at which the role assignment ceases to be active. A value of `DateTIme.MaxValue` means the role assignment never expires. The time must be later than 1/1/1970 00:00.

### Exceptions

`EndTime` throws an `ArgumentExceptIon` if you try to set the end time earlier than the start time or earlier than 2400 hours UTC, 1 January, 1970.

## Id

Gets the GUID of the role assignment.

### Syntax

```
Guid Id {get;}
```

### Property value

The GUID of the role assignment.



## IsRoleOrphaned

Indicates whether the role assignment is orphaned due to a missing or invalid role.

### Syntax

```
bool IsRoleOrphaned {get;}
```

### Property value

Returns true if this role assignment is an orphan.

## IsTrusteeOrphaned

Indicates whether the role assignment is orphaned due to a missing or invalid user or group.

### Syntax

```
bool IsTrusteeOrphaned {get;}
```

### Property value

Returns true if this role assignment is an orphan.

## LocalTrustee

Gets or sets the local user or group being assigned.

### Syntax

```
string LocalTrustee {get; set;}
```

### Property value

The local trustee; either a group or user. You can set either a name or ID for a local user. A local group string begins with the type flag %. You cannot specify a null or empty string. A user or group name string must match the regular expression (Regex):

```
@'^%?[\.a-zA-Z0-9_-]+\$?@localhost$"
```



A user ID string must match the regular expression:

```
@'^#[0-9]+@localhost$"
```

## Exceptions

LocalTrustee may throw one of the following exceptions:

- `ArgumentException` if the local trustee string is `null` or empty.
- `ApplicationException` if the method fails to update the role assignment (see the message returned by the exception for the reason).

## Role

Gets or sets the role.

### Syntax

```
IRole Role {get; set;}
```

### Property value

The object representing the role.

## Exceptions

`Role` throws an `ApplicationException` if the role you specify is not in the current or parent zone.

If the `Role` property fails, see the message returned by the exception for more specific information about the reason for the failure.

## StartTime

Gets or sets the time from which this role assignment becomes effective.

### Syntax

```
DateTime StartTime {get; set;}
```



## Property value

The time at which the role assignment becomes active. A value of `DateTime.MinValue` means the role becomes effective immediately. The time must be later than 1/1/1970 00:00.

## Exceptions

`StartTime` throws an `ArgumentException` if you try to set the start time later than the end time or earlier than 2400 hours UTC, 1 January, 1970.

## TrusteeDn

Gets or sets the user associated with the role.

## Syntax

```
string TrusteeDn {get; set;}
```

## Property value

The distinguished name of the user.

## Exceptions

`TrusteeDn` may throw one of the following exceptions:

- `ArgumentException` if the trustee string is null or empty.
- `ApplicationException` if the method fails to update the role assignment trustee (see the message returned by the exception for the reason).

## TrusteeType

Gets or sets the trustee type of the role assignment.

## Syntax

```
TrusteeType TrusteeType {get; set;}
```

## Property value

The type of trustee.



Possible values:

```
public enum TrusteeType
{
    // Unknown
    Unknown = 0,
    // AD User
    User = 1,
    // AD group
    Group = 2,
    // Local UNIX user
    LocalUser = 4,
    // Local UNIX group
    LocalGroup = 8,
    // All AD Users
    AllADUsers = 16,
    // All local UNIX accounts
    AllUnixUser = 32
    // Local windows user
    LocalWindowsUser = 64
    // Local windows group
    LocalWindowsGroup = 128
    // All local windows users
    AllWindowsUsers = 256
};
```

## Exceptions

TrusteeType throws an `ArgumentException` if you try to set the trustee type to any value other than `AllADUsers` or `AllUnixUser`.

## RoleAssignments

The `RoleAssignments` class manages a collection of Role Assignment objects.

## Syntax

```
public interface IRoleAssignments
```

## Methods

The `RoleAssignments` class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Returns an enumeration of role assignments.



## GetEnumerator

Returns an enumeration of role assignments.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

The set of `RoleAssignment` objects.

### Exceptions

`GetEnumerator` throws an `ApplicationException` if it cannot find the scope in the zone, cannot find the role, or cannot find authorization data for the zone.

If the `GetEnumerator` method fails, see the message returned by the exception for more specific information about the reason for the failure.

## Roles

The `Roles` class manages a collection of `Role` objects.

### Syntax

```
public interface IRoles
```

### Methods

The `Roles` class provides the following method:

<b>This method</b>	<b>Does this</b>
<code>GetEnumerator</code>	Returns an enumeration of roles.

## GetEnumerator

Returns an enumeration of roles.



## Syntax

```
IEnumerator GetEnumerator()
```

## Return value

The set of [Role](#) objects.

## Exceptions

`GetEnumerator` throws an `ApplicationException` if it cannot find authorization data for the zone.

If the `GetEnumerator` method fails, see the message returned by the exception for more specific information about the reason for the failure.

# Ssh

Represents an SSH application access right.

## Syntax

```
public interface ISsh: IRight
```

## Discussion

An SSH (Secure Shell) application right gives a user the ability to access the authorized SSH-enabled application.

## Methods

The `Ssh` class provides the following methods:

This method	Does this
<code>Commit</code>	Commits changes in the right to Active Directory. (Inherited from <a href="#">Right</a> .)
<code>Delete</code>	Deletes the right. (Inherited from <a href="#">Right</a> .)



## Properties

The `Ssh` class provides the following properties:

This property	Does this
<code>Application</code>	Gets or sets the SSH application.
<code>Description</code>	Gets or sets the description of the right. (Inherited from <code>Right</code> .)
<code>IsReadable</code>	Indicates whether the right is readable. (Inherited from <code>Right</code> .)
<code>IsWritable</code>	Indicates whether the right is writable. (Inherited from <code>Right</code> .)
<code>Name</code>	Gets or sets the name of the right. (Inherited from <code>Right</code> .)
<code>Zone</code>	Gets the zone this right belongs to. (Inherited from <code>Right</code> .)

## Application

Gets or sets the SSH application for which this is a right.

### Syntax

```
string Application {get; set;}
```

### Property value

The file path of the application.

### Exceptions

`Application` throws an `ArgumentException` if the application string is `null` or empty.

## Sshs

The `Sshs` class manages a collection of SSH application access rights.



## Syntax

```
public interface ISshs
```

## Methods

The Sshs class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Returns the enumerator you can use to enumerate all SSH rights.

## GetEnumerator

Returns an enumeration of SSH access rights.

## Syntax

```
IEnumerator GetEnumerator()
```

## Return value

An enumerator you can use to list all the Ssh objects.

## User

The User class enables Centrify to associate existing Active Directory user accounts with UNIX profiles that contain the attributes required for users to log on to UNIX computers.

## Syntax

```
public interface IUser
```



## Discussion

These additional UNIX-specific attributes that make up the UNIX profile for an Active Directory user are stored and managed within the `UserUnixProfile` object.

## Methods

The `User` class provides the following methods:

This method	Does this
<code>AddUnixProfile</code>	Adds a new UNIX profile for a user to the specified zone.
<code>Commit</code>	Commits the changes to the <code>User</code> object and saves them in Active Directory.
<code>CommitWithoutCheck</code>	Commits the changes to the <code>User</code> object without validating any of the data before saving.
<code>GetDirectoryEntry</code>	Returns an instance of the <code>DirectoryEntry</code> for the user from Active Directory.
<code>GetRoleAssignmentsFromDomain</code>	Returns the collection of all role assignments for a user in a specified domain.
<code>GetRoleAssignmentsFromForest</code>	Returns the collection of all role assignments for a user in a specified forest.
<code>Refresh</code>	Reloads the data in the cache from Active Directory.

## Properties

The `User` class provides the following properties:

This property	Does this
<code>AdsiInterface</code>	Gets the ADSI interface of the user object in Active Directory.
<code>ADsPath</code>	Gets the LDAP path to the Active Directory user object.
<code>ID</code>	Gets the UID for the user as a string.
<code>UnixProfiles</code>	Gets the collection of UNIX profiles for the user.



## AddUnixProfile

Adds a new UNIX profile for an existing Active Directory user account to the specified zone.

### Syntax

```
IUserUnixProfile AddUnixProfile (IZone zone, int uid, string name, string shell, string homeDir, int primaryGroup)
```

```
IUserUnixProfile AddUnixProfile (IZone zone, long uid, string name, string shell, string homeDir, long primaryGroup)
```

### Parameters

Specify the following parameters when using this method.

Parameter	Description
zone	The destination zone for the new user information.
uid	The UID of the user associated with the profile.
name	The UNIX login name of the user associated with the profile.
shell	The default shell of the user associated with the profile.
homeDir	The default home directory of the user associated with the profile.
primaryGroup	The GID of the primary group of the user associated with the profile.

### Return value

A new `UserUnixProfile` object.

### Discussion

There are two versions of this method: one designed for COM-based programs that supports a 32-bit signed number for the `uid` and `primaryGroup` arguments and one designed for .NET-based programs that allows a 64-bit signed number for the arguments.

### Exceptions

`AddUnixProfile` throws a `NotSupportedException` if the zone schema is not supported.

### Example

The following code sample illustrates using `AddUnixProfile` in a script:

• • • • •

```
...
// Create a CIMS object to interact with AD
ICims cims = new Cims();
// Note the lack of the cims.connect function.
// By default, this application will use the connection to the
domain controller
// and existing credentials from the computer already logged in.
// Get the user object
IUser objUser = cims.GetUserByPath(strUser);
// Get the zone object
IZone objZone = cims.GetZoneByPath("cn=" + strZone + "," +
strContainerDN);
IUserUnixProfile objUserUnixProfile;
if (objUser.UnixProfiles.Find(objZone) == null)
{
    //New user for the zone
    long lngUID = objZone.NextUID;
    string strShell = objZone.DefaultShell;
    string strHome = objZone.DefaultHomeDirectory;
    if (bool.Parse(bPrivate))
    {
        // Create the user as a member of a private group
        objUserUnixProfile = objUser.AddUnixProfile(objZone,
lngUID, strUnixAccount,
                strShell, strHome, lngUID);
    }
    else
    {
        // Create the user as a member of the default group
        IGroupUnixProfile objDefaultGroup = objZone.DefaultGroup;
        long lngGID = 10000; // use 10000 as default
        if (objDefaultGroup != null)
        {
            lngGID = objDefaultGroup.GroupId;
        }
        objUserUnixProfile = objUser.AddUnixProfile(objZone,
lngUID, strUnixAccount,
                strShell, strHome, lngGID);
    }
    // Enable the UNIX profile for the end user
    objUserUnixProfile.UnixEnabled = true;
    objUserUnixProfile.Commit();
}
else
{
    Console.WriteLine(strUser + " is already a member of " +
strZone);
    return;
}
Console.WriteLine("User " + strUser + " was successfully added to
zone " + strZone + ".");
...

```



## Commit

Commits any changes or updates to the User object and saves the changes to Active Directory.

### Syntax

```
void Commit()
```

### Discussion

When you use this method, it checks and validates the data before saving it in Active Directory.

### Exceptions

Commit may throw one of the following exceptions:

- `ApplicationException` if any field in the UNIX profile is not valid.
- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.
- `UnauthorizedAccessException` if you have insufficient access rights to commit changes to the Active Directory object.

### Example

The following code sample illustrates using Commit in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the user object  
set objUser = cims.GetUserByPath("LDAP://CN=pat.hu,CN=Users,  
DC=ajax,DC=org")  
'Add the UNIX profile for the user  
set objUserUnixProfile = objUser.AddUnixProfile(objZone, 623,  
"pat_hu", "/bin/bash", "/home/pat_hu", 623)  
'Enable the user's UNIX profile  
objUserUnixProfile.UnixEnabled = True  
'Update Active Directory  
objUserUnixProfile.commit  
...
```



## CommitWithoutCheck

Commits any changes or updates to the User object and saves the changes to Active Directory.

### Syntax

```
void CommitWithoutCheck()
```

### Discussion

When you use this method, it does not validate any of the data before saving.

### Exceptions

CommitWithoutCheck may throw one of the following exceptions:

- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.
- `UnauthorizedAccessException` if you have insufficient access rights to commit changes on the Active Directory object.

### Example

The following code sample illustrates using `CommitWithoutCheck` in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the user object  
set objUser = cims.GetUserByPath("LDAP://CN=pat.hu,CN=Users,  
DC=ajax,DC=org")  
'Add the UNIX profile for the user  
set objUserUnixProfile = objUser.AddUnixProfile(objZone, nextuid,  
unixlogin, defaultshell, homedir, admingid)  
'Enable the user's UNIX profile  
objUserUnixProfile.UnixEnabled = True  
'Update Active Directory without validating the UNIX profile  
objUserUnixProfile.CommitWithoutCheck  
...
```

## GetDirectoryEntry

Returns the directory entry for the user from Active Directory.



## Syntax

```
DirectoryEntry GetDirectoryEntry()
```

## Return value

A directory entry for the service connection point that represents the user's UNIX profile.

## Discussion

The `DirectoryEntry` object represents the directory object for the user and its associated attributes.

**Note:** This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

## Exceptions

`GetDirectoryEntry` throws an `ApplicationException` if it cannot get the directory object.

## GetRoleAssignmentsFromDomain

Returns the collection of all role assignments associated with a user in a specified domain.

## Syntax

```
IRoleAssignments GetRoleAssignmentsFromDomain(string domain)
```

## Parameters

Specify the following parameter when using this method:

Parameter	Description
<code>domain</code>	The domain to search for the user's role assignments.

## Return value

A collection of role assignment objects representing all of the role assignments explicitly assigned to this user in the specified domain or in the currently joined



domain.

## Discussion

This method only returns the role assignments that have been explicitly assigned to the user. The method will look for stored credentials to access the specified domain. If there are no stored credentials, the method uses the default credentials for the current user.

If you don't specify a domain by passing an empty string ("") to the method, the method returns role assignments from the currently joined domain.

## Example

The following code sample illustrates using `GetRoleAssignmentsFromDomain` in a script:

```
...  
# New Cims object  
$cims = New-Object ("Centrify.DirectControl.API.Cims");  
# Get IUser object  
$objUserDn = "CN=user1,CN=Users,DC=domain,DC=com";  
$objUser = $cims.GetUser($objUserDn);  
# Get role assignments from domain  
$objUser.GetRoleAssignmentsFromDomain("domain.com")  
...
```

## GetRoleAssignmentsFromForest

Returns the collection of all role assignments associated with a user in a specified Active Directory forest.

## Syntax

```
IRoleAssignments GetRoleAssignmentsFromForest(string forest)
```

## Parameters

Specify the following parameter when using this method:

Parameter	Description
forest	The forest to search for the user's role assignments.



## Return value

A collection of role assignment objects representing all of the role assignments explicitly assigned to this user in the specified forest or in the currently joined forest.

## Discussion

This method only returns the role assignments that have been explicitly assigned to the user. The method will look for stored credentials to access the specified forest. If there are no stored credentials, the method uses the default credentials for the current user.

If you don't specify a forest by passing an empty string ("" ) to the method, the method returns role assignments from the currently joined forest.

## Example

The following code sample illustrates using `GetRoleAssignmentsFromForest` in a script:

```
...
# New Cims object
$cims = New-Object ("Centrify.DirectControl.API.Cims");
# Get IUser object
$objUserDn = "CN=user1,CN=Users,DC=domain,DC=com";
$objUser = $cims.GetUser($objUserDn);
# Get role assignments from forest
$objUser.GetRoleAssignmentsFromForest("forest.com")
...
```

## Refresh

Reloads the User object data from the data in Active Directory.

## Syntax

```
void Refresh()
```

## Discussion

This method refreshes the user information in the cached object to ensure it is synchronized with the latest information in Active Directory.



## Exceptions

Refresh throws a `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using Refresh in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath
("LDAP://CN=corporate,CN=zones,CN=centrify,CN=program
data,DC=sierra,DC=com")
'Get the user object
set objUser = cims.GetUserByPath
("LDAP://CN=pat.hu,CN=Users,DC=ajax,DC=org")
'Get the UNIX profile for the user
profile = objUser.UnixProfiles
'Enable the user's UNIX profile
profile.UnixEnabled = True
'Reload the user object
objUser.Refresh
...
```

## AdsInterface

Gets the ADSI interface of the user object in Active Directory.

## Syntax

```
IADsUser AdsInterface {get;}
```

## Property value

The ADSI interface of the user object in Active Directory.

## Example

The following code sample illustrates using `AdsInterface` in a script:

```
...
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Get the UNIX profile for the user
```

• • • • •

```
profile = objUser.UnixProfiles
'Display the ADSI interface for the user
wScript.Echo "ADSI: " & profile.AdsiInterface
...
```

## ADsPath

Gets the LDAP path to the Active Directory user object.

### Syntax

```
string ADsPath {get;}
```

### Property value

The LDAP path to the Active Directory user object.

### Discussion

The basic format for the LDAP path is:

```
LDAP://[<domain>/]<attr>=<name>...,dc=<domain part>...
```

For example, if the user object is john.doe in the organizational unit consultants and the domain is centri fy . com, the LDAP path to the object looks like this:

```
LDAP://cn=john.doe,ou=consultants,dc=centri fy ,dc=com
```

### Example

The following code sample illustrates using ADsPath in a script:

```
...
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Get the UNIX profile for the user
profile = objUser.UnixProfiles
'Display the LDAP path for the user
wScript.Echo "LDAP Path: " & profile.ADsPath
...
```

## ID

Gets the unique identifier for the user as a string value.

• • • • •

## Syntax

```
string ID {get;}
```

## Property value

The unique identifier for the user as a string.

## Example

The following code sample illustrates using ID in a script:

```
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Get the UNIX profile for the user
profile = objUser.UnixProfiles
'Display the UID for the user
wscript.Echo "User Identifier (UID): " & profile.ID
...
```

## UnixProfiles

Gets all of the UNIX profiles for a specified Active Directory user in the current domain.

## Syntax

```
IUserUnixProfiles UnixProfiles {get;}
```

## Property value

The collection of UNIX profiles for the user.

## Discussion

The resulting object, `UserUnixProfiles`, is the collection of UNIX profiles that have been defined for the user across all zones.

## Example

The following code sample illustrates using `unixProfiles` in a script:

```
...'
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Get the user object
```



```
set objUser = cims.GetUserByPath("LDAP://CN=tai.wu,CN=Users,DC=ajax,DC=org")
'Look up the user's UNIX profile in the zone
dim objUserUnixProfiles
set objUserUnixProfiles = objUser.UnixProfiles
set objUserUnixProfile = objUserUnixProfiles.Find(objZone)
```

## UserUnixProfile

The `UserUnixProfile` class is an information class for managing user information in a specific zone.

### Syntax

```
public interface IUserUnixProfile
```

### Discussion

A user's zone-specific UNIX profile includes the numeric UID value, numeric GID value, default login shell, and default home directory. The GID can be associated with a standard Active Directory group, a standalone UNIX-only group profile not associated with any Active Directory group, and a local user profile.

### Methods

The `UserUnixProfile` class provides the following methods:

This method	Does this
<code>Commit</code>	Commits changes to the user profile to Active Directory.
<code>Delete</code>	Marks the user profile for deletion from Active Directory.
<code>GetDirectoryEntry</code>	Returns the directory entry for the UNIX user profile from Active Directory.
<code>GetPrimaryGroup</code>	Returns the UNIX profile of the primary group of the user.
<code>Refresh</code>	Reloads cached object data from Active Directory.
<code>Validate</code>	Checks whether the user profile contains valid data and can be committed to Active Directory.



## Properties

The `UserUnixProfile` class provides the following properties:

This property	Does this
<code>ADSPATH</code>	Gets the LDAP path to the UNIX data object.
<code>Cims</code>	Gets the Cims object for the user.
<code>HomeDirectory</code>	Gets or sets the home directory for the user.
<code>ID</code>	Gets the unique identifier for the <code>UserUnixProfile</code> data object.
<code>IsForeign</code>	Indicates whether the Active Directory user associated with a UNIX profile is defined in a different forest than the zone (not applicable to local user profiles).
<code>IsOrphan</code>	Indicates whether this UNIX user is not associated with a corresponding Active Directory user (not applicable to local user profiles).
<code>IsReadable</code>	Indicates whether the Active Directory object is readable.
<code>IsSFU</code>	Indicates whether this UNIX user is an SFU zone profile (not applicable to local user profiles).
<code>IsWritable</code>	Determines whether the Active Directory object is writable.
<code>Name</code>	Gets or sets the UNIX login name of the user.
<code>PrimaryGroup</code>	Gets or sets the GID of the user's primary group.
<code>ProfileState</code>	Gets or sets the profile state of the local user profile (local user profiles only).
<code>Shell</code>	Gets or sets the default shell for the user.
<code>Type</code>	Gets the <code>UserUnixProfile</code> type for the user.
<code>UnixEnabled</code>	Determines whether the user's UNIX profile is enabled for access to the zone.
<code>User</code>	Gets the user object associated with this user UNIX profile (not applicable to local user profiles).
<code>UserId</code>	Gets or sets the UNIX user identifier (UID).
<code>Zone</code>	Gets the zone object associated with the user.

## Commit

Commits changes to the user profile object and saves the changes in Active Directory.

## Syntax

```
void Commit()
```



## Discussion

If you are creating a new UNIX profile or updating a user's primary group or other attributes, you must use this method to complete the operation. If you have marked an object for deletion, this method deletes the object from Active Directory.

## Exceptions

`Commit` may throw one of the following exceptions:

- `UnauthorizedAccessException` if your permissions are not sufficient to commit the Active Directory data object.
- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using `Commit` in a script:

```
...
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Set the UNIX profile for the user
set profile = objUser.SetUnixProfile(objZone, 10001, "pat",
"/bin/bash", "/home/pat", 10001)
'validate the user's UNIX profile
profile.Validate
profile.Commit
...
```

## Delete

Marks the user profile object for deletion from Active Directory.

## Syntax

```
void Delete()
```



## Discussion

This method does not delete the user profile. After you mark an object for deletion, you must use the **Commit** method to commit changes to the user object to Active Directory. When the Commit method is executed, the UNIX profile is deleted from Active Directory to complete the operation.

## Example

The following code sample illustrates using `Delete` in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the Active Directory user object  
set objUser = cims.GetUser("ajax.org/Users/pat.hu")  
'Get the UNIX profile for the user  
profile = objUser.UnixProfiles  
'Mark the user profile for deletion  
profile.Delete  
...
```

## GetDirectoryEntry

Returns an instance of the directory entry for the user's UNIX profile from Active Directory.

### Syntax

```
DirectoryEntry GetDirectoryEntry()
```

### Return value

A directory entry for the service connection point that represents the user's UNIX profile in the zone.

## Discussion

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

• • • • •

## GetPrimaryGroup

Returns the UNIX profile of the primary group of the user.

### Syntax

```
IGroupUnixProfile GetPrimaryGroup();
```

### Return value

The UNIX profile of the user's primary group.

### Example

The following code sample illustrates using `GetPrimaryGroup` in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the Active Directory user object  
set objUser = cims.GetUser("ajax.org/Users/pat.hu")  
'Get the UNIX profile for the user  
profile = objUser.UnixProfiles  
'Display the LDAP path for the user  
wScript.Echo "Primary GID: " & profile.GetPrimaryGroup().GID  
...
```

## Refresh

Reloads UNIX profile data from the data in Active Directory.

### Syntax

```
void Refresh()
```

### Discussion

This method refreshes the UNIX profile information in the cached object to ensure it is synchronized with the latest information in Active Directory.

### Example

The following code sample illustrates using `Refresh` in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
```

• • • • •

```
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Get the UNIX profile for the user
profile = objUser.UnixProfiles
'Reload the user's UNIX profile
profile.Refresh
...
```

## Validate

Checks whether the user profile contains valid data and can be committed to Active Directory.

## Syntax

```
void validate()
```

## Discussion

This method checks for errors in the UNIX profile fields and verifies that the profile includes a valid primary group, user name, UID, home directory, shell, and zone type. The method also verifies that the entry is not a duplicate of any existing profile in the zone. The method does not perform any of these checks, however, if the user profile is marked for deletion or if the profile has not been modified.

## Exceptions

`validate` throws an `ApplicationException` if the UNIX profile is missing data or contains invalid data.

## Example

The following code sample illustrates using `validate` in a script:

```
...
'Get the zone object
set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Set the UNIX profile for the user
set profile = objUser.SetUnixProfile(objZone, 10001, "pat",
"/bin/bash", "/home/pat", 10001)
'Validate the user's UNIX profile
profile.Validate
...
```

• • • • •

## ADsPath

Gets the LDAP path to the UNIX profile data object.

### Syntax

```
string ADsPath {get;}
```

### Property value

The LDAP path to the UNIX profile data object.

### Example

The following code sample illustrates using ADsPath in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the Active Directory user object  
set objUser = cims.GetUser("ajax.org/Users/pat.hu")  
'Get the UNIX profile for the user  
profile = objUser.UnixProfileByUid(10001)  
'Display the LDAP for the user's UNIX profile  
wScript.Echo "LDAP Path: " & profile.ADsPath  
...
```

## Cims

Gets the Cims object for the user.

### Syntax

```
Cims Cims {get;}
```

### Property value

The Cims object.

### Discussion

This property serves as a shortcut for retrieving data.

• • • • •

## HomeDirectory

Gets or sets the home directory for the user.

### Syntax

```
string HomeDirectory {get; set;}
```

### Property value

A string that defines the path to the default home directory for the user from the user's UNIX profile.

### Example

The following code sample illustrates using HomeDirectory in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the Active Directory user object  
set objUser = cims.GetUser("ajax.org/Users/pat.hu")  
'Get the UNIX profile for the user  
profile = objUser.UnixProfileByUid(10001)  
'Change the default home directory in the user's UNIX profile  
set profile.HomeDirectory = "/home/all_users/pathu"  
...
```

## ID

Gets the unique identifier for the [UserUnixProfile](#) data object.

### Syntax

```
string ID {get;}
```

### Property value

The unique identifier for the [UserUnixProfile](#) data object.

### Example

The following code sample illustrates using ID in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
```

• • • • •

```
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Get the UNIX profile for the user
profile = objUser.UnixProfileByUid(10001)
'Display the unique identifier for the user's UNIX profile
wScript.Echo "Unique ID: " & profile.ID
...
```

## IsForeign

Indicates whether the corresponding Active Directory user for a UNIX profile is in a different Active Directory forest than the forest associated with the user's UNIX profile in the zone.

### Syntax

```
bool IsForeign {get;}
```

### Property value

Returns true if the UNIX profile is associated with an Active Directory user in a different forest.

### Discussion

If the Active Directory user is in a different forest than the one associated with a top-level Cims object, the property returns true.

### Example

The following code sample illustrates using IsForeign in a script:

```
...
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Check the forest for Users in the zone
For each profile in objZone.GetUserUnixProfiles
if profile.IsForeign then
    wScript.Echo "Foreign user: " & profile.Name
end if
next
...
```



## IsOrphan

Indicates whether this UNIX user is not associated with a corresponding Active Directory user.

### Syntax

```
bool IsOrphan {get;}
```

### Property value

Returns `true` if the corresponding Active Directory user for a UNIX profile is not found, or `false` if the corresponding Active Directory user for the UNIX profile exists.

### Discussion

This property can be used to determine whether the Active Directory user associated with a UNIX profile has been deleted from Active Directory.

### Example

The following code sample illustrates using `IsOrphan` in a script:

```
...
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Check for Orphan Users in the zone
For each profile in objZone.GetUserUnixProfiles
if profile.IsOrphan then
    wscript.Echo "Orphan user: " & profile.Name
end if
next
...
```

## IsReadable

Indicates whether the user profile object in Active Directory is readable for the current user credentials.

### Syntax

```
bool IsReadable {get;}
```

• • • • •

## Property value

Returns true if the `UserUnixProfile` object is readable, or false if the object is not readable.

## Discussion

This property returns a value of true if the user accessing the user profile object in Active Directory has sufficient permissions to read its properties.

## Example

The following code sample illustrates using `IsReadable` in a script:

```
...
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Get the UNIX profile for the user
profile = objUser.UnixProfileByUid(10001)
'Check whether the user's UNIX profile is readable
if not profile.IsReadable then
    wScript.Echo "No permission to read the UNIX profile!"
else
    wScript.Echo "UNIX login name: " & profile.Name
end if
...
```

## IsSFU

Indicates whether the UNIX user profile is in a Services for UNIX (SFU) zone.

## Syntax

```
bool IsSFU {get;}
```

## Property value

Returns true if the user profile is a Services for UNIX (SFU) profile.

## IsWritable

Indicates whether the user profile object is writable for the current user's credentials.

• • • • •

## Syntax

```
bool Iswritable {get;}
```

## Property value

Returns `true` if the `UnixUserProfile` object is writable, or `false` if the object is not writable.

## Discussion

This property returns a value of `true` if the user accessing the user profile object in Active Directory has sufficient permissions to change the user profile object's properties.

## Example

The following code sample illustrates using `IsWritable` in a script:

```
...
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Get the UNIX profile for the user
profile = objUser.UnixProfileByUid(10001)
'Check whether the user's UNIX profile is writable
if not profile.IsWritable then
    wScript.Echo "No permission to change the UNIX profile!"
end if
...
```

## Name

Gets or sets the UNIX login name of the user in the user's UNIX profile.

## Syntax

```
string Name {get; set;}
```

## Property value

The UNIX login name from the user's UNIX profile.

## Example

The following code sample illustrates using `Name` in a script:

• • • • •

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the Active Directory user object  
set objUser = cims.GetUser("ajax.org/Users/pat.hu")  
'Set the UNIX profile for the user  
set profile = objUser.SetUnixProfile(objZone, 10001, "pat",  
"/bin/bash", "/home/pat", 10001)  
'Display the user's UNIX login name  
profile.Name  
...
```

## PrimaryGroup

Gets or sets the group identifier (GID) of the primary group for the user.

### Syntax

```
Long PrimaryGroup {get; set;}
```

### Property value

The value used as the GID for the user's primary group.

### Discussion

This method is used internally by .NET modules.

### Example

The following code sample illustrates using `PrimaryGroup` in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the Active Directory user object  
set objUser = cims.GetUser("ajax.org/Users/pat.hu")  
'Set the GID for the user's primary group  
Set objUser.PrimaryGroup = 490007  
...
```

## ProfileState

Gets the profile state of an existing local user or sets the profile of the specified local user.

• • • • •

## Syntax

```
UserProfileState ProfileState{get; set;}
```

## Property value

The profile state of the specified local user.

## Exceptions

`ProfileState` throws an `InvalidOperationException` if the user you specify is not a local user.

## Shell

Gets or sets the default shell for the user.

## Syntax

```
string shell {get; set;}
```

## Property value

A string that defines the path to the default shell for the user from the user's UNIX profile.

## Example

The following code sample illustrates using `Shell` in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the Active Directory user object  
set objUser = cims.GetUser("ajax.org/Users/pat.hu")  
'Get the UNIX profile for the user  
profile = objUser.UnixProfileByUid(10001)  
'Change the default shell in the user's UNIX profile  
set profile.Shell = "bin/sh"  
...
```

## Type

Gets the user UNIX profile type for the user.

• • • • •

## Syntax

```
UserUnixProfileType Type {get;}
```

## Property value

A numeric value that indicates whether the UNIX profile is a standard Centrify profile or a Service for UNIX (SFU) profile.

Possible values:

```
public enum UserUnixProfileType
{
    // Centrify user
    Centrify = 0,
    // Microsoft Service for Unix type
    Sfu = 1,
    // MIT Kerberos-realm trusted user
    MIT = 2
}
```

## Discussion

There is no AD User object corresponding to an MIT user type of profile.

## Example

The following code sample illustrates using Type in a script:

```
...
'Get the zone object
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")
'Get the Active Directory user object
set objUser = cims.GetUser("ajax.org/Users/pat.hu")
'Get the UNIX profile for the user
profile = objUser.UnixProfileByUid(10001)
'Check the profile type in the user's UNIX profile
if profile.Type = 0
    wScript.Echo "Standard UNIX profile"
else
    wScript.Echo "SFU user UNIX profile"
end if
...
```

## UnixEnabled

Determines whether the user's UNIX profile is enabled for access to the zone. This attribute is only applicable in classic zones and for backwards compatibility.

• • • • •

## Syntax

```
bool UnixEnabled {get; set;}
```

## Property value

Returns `true` if the user's UNIX profile is enabled for access in a classic zone, or `false` if the UNIX profile is not enabled for access to the zone.

## Discussion

The `unixEnabled` attribute determines whether a specific profile is enabled or disabled for access to the zone. If this property is set to `true`, the user's UNIX profile can be used to access the current zone. If this property is set `false`, the user cannot log on to computers in the zone using the disabled UNIX profile.

## Exceptions

`UnixEnabled` throws an `InvalidOperationException` if the user is assigned to a hierarchical zone.

## Example

The following code sample illustrates using `UnixEnabled` in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the user object  
set objUser = cims.GetUserByPath  
("LDAP://CN=pat.hu,CN=Users,DC=ajax,DC=org")  
'Disable the user's UNIX profile in this zone  
objUserUnixProfile.UnixEnabled = False  
...
```

## User

Gets the Active Directory user object associated with this user UNIX profile.

## Syntax

```
IUser User {get;}
```

## Property value

The user object.

• • • • •

## UserId

Gets or sets the UID of the user.

### Syntax

```
long UserId {get; set;}
```

### Property value

The UID of the user.

## Zone

Gets the zone object associated with the user.

### Syntax

```
IZone Zone {get;}
```

### Property value

The zone object associated with the user.

## Example

The following code sample illustrates using Zone in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
'Get the Active Directory user object  
set objUser = cims.GetUser("ajax.org/Users/pat.hu")  
'Get the UNIX profile for the user  
profile = objUser.UnixProfileByUid(10001)  
'Display the zone associated with the user's UNIX profile  
wScript.Echo "Zone: " & profile.Zone  
...
```

## UserUnixProfiles

The UserUnixProfiles class is used to manage a collection of user profiles.



## Syntax

```
public interface IUserUnixProfiles
```

## Discussion

The collection of user profiles contained in the object depend on how the object was obtained:

- When you call `user.UnixProfiles`, the `UserUnixProfiles` object returned enumerates all of the profiles defined for a specific Active Directory user across all zones in the current domain.
- When you call `zone.GetUserUnixProfiles`, the `UserUnixProfiles` object returned enumerates all of the profiles defined for a specific Active Directory user in a specific zone.

## Methods

The `UserUnixProfiles` class provides the following methods:

This method	Does this
<code>GetEnumerator</code>	Returns an enumeration of user profiles.
<code>Refresh</code>	Reloads the collection of UNIX profiles from Active Directory.

## Properties

The `UserUnixProfiles` class provides the following properties:

This property	Does this
<code>Count</code>	Gets the number of UNIX profiles defined in the collection of <code>UserUnixProfiles</code> .
<code>IsEmpty</code>	Indicates whether the <code>UserUnixProfiles</code> object contains any profiles.

## GetEnumerator

Returns an enumeration of user profiles.

• • • • •

## Syntax

```
IEnumerator GetEnumerator()
```

## Return value

The set of user profile objects.

## Refresh

Reloads the collection of UNIX profiles from Active Directory.

## Syntax

```
void Refresh()
```

## Discussion

This method refreshes the collections of UNIX profiles in the cached object to ensure the object is synchronized with the latest information in Active Directory.

## Count

Determines the total number of UNIX profiles defined in the `UserUnixProfiles` collection.

## Syntax

```
int Count {get;}
```

## Property value

The number of UNIX profiles in the set.

## Example

The following code sample illustrates using `Count` in a script:

```
...  
'Get the zone object  
Set objZone = cims.GetZone("ajax.org/UNIX/Zones/pilot")  
Set objUserUnixProfiles = objUser.UnixProfiles  
wScript.Echo "Profile count: " & objUserUnixProfiles.Count  
...
```



## IsEmpty

Determines whether the collection of UNIX user profiles is empty.

### Syntax

```
bool IsEmpty {get;}
```

### Property value

Returns true if there are no user profiles in the `UserUnixProfiles` object.

### Discussion

Unlike the `Count` property, the `IsEmpty` property does not query all of the profiles in the collection before it returns a value. If you only need to determine whether any profiles are defined, you should call this property for a faster response.

## WindowsApplication

This class represents a Windows application right.

### Syntax

```
public interface IWindowsApplication:IRight
```

### Methods

The `WindowsApplication` class provides the following methods:

This method	Does this
<code>Commit</code>	Commits changes in the right to Active Directory. (Inherited from <code>Right</code> .)
<code>CreateApplicationCriteria</code>	Creates the match criteria used to identify the Windows application to which you want to grant a right.
<code>Delete</code>	Removes the right. (Inherited from <code>Right</code> .)



## Properties

The `WindowsApplication` class provides the following properties:

This property	Does this
<code>ApplicationCriteriaList</code>	Gets or sets the properties that are used to identify a specific Windows application.
<code>Description</code>	Gets or sets the description of the right. (Inherited from <code>Right</code> .)
<code>IsReadable</code>	Indicates whether the right is readable. (Inherited from <code>Right</code> .)
<code>IsWritable</code>	Indicates whether the right is writable. (Inherited from <code>Right</code> .)
<code>Name</code>	Gets or sets the name of the right. (Inherited from <code>Right</code> .)
<code>AddLocalGroupPartialProfile</code>	Gets or sets the priority of this right.
<code>RequirePassword</code>	Gets or sets whether the user's password is required when this right is used.
<code>RunAs</code>	Gets or sets the SID for the run-as user or an SID for the user assigned the right (VBScript).
<code>RunAsList</code>	Gets or sets the SID for the run-as user or a list of SIDs for the users assigned the right (.NET).
<code>RunAsType</code>	Gets or sets the run-as type for the right.
<code>Zone</code>	Gets the zone this right belongs to. (Inherited from <code>Right</code> .)

## Discussion

A Windows application right enables a user to run a Windows application with the privileges of another user or as a member of an Active Directory or built-in group. For example, you can use a Windows application right to give a standard Windows user elevated privileges to run a database management application as a database administrator.



## CreateApplicationCriteria

Creates the match criteria to identify the Windows application and related requirements to which you want to grant a right.

### Syntax

```
void CreateApplicationRight()
```

### Discussion

This method initiates the collection of the set of match criteria, defined using the properties of the `WindowsApplicationCriteria` class, to identify a specific Windows application to which you want to grant a right.

### Example

See [ApplicationCriteriaList](#) for a code sample that illustrates using `CreateApplicationRight` in a script.

## ApplicationCriteriaList

Gets or sets the list of properties that are used to identify a specific Windows application.

### Syntax

In .NET:

```
IEnumerable<IWindowsApplicationCriteria> ApplicationCriteriaList  
{get; set;}
```

In VBScript:

```
object[] ApplicationCriteriaList {get; set;}
```

### Property value

The complete match criteria defined to identify a specific Windows application.

### Example

The following code sample illustrates using `ApplicationCriteriaList` in a script:



```
// Create a new windows application right with some basic
properties.
$objWindowsApplication = $objZone.CreateWindowsApplication();
    $objWindowsApplication.Name = $strWindowsApplication;
    $objWindowsApplication.RunAsType = $runAsType;
    $objWindowsApplication.RunAsString = $strDnList;
    $objWindowsApplication.RequirePassword = $requirePassword;
    $objWindowsApplication.Description = "optional description";
    $objWindowsApplication.Priority = 0;
// Specify the criteria used to identify the windows application.
$listType = ("System.Collections.Generic.List`1" -as
"type");
    $listType = $listType.MakeGenericType( @(
("Centrify.DirectControl.API.IWindowsApplicationCriteria" -as
"type")));
    $criteriaList = [Activator]::CreateInstance($listType);
    $objApplicationCriteria =
$objWindowsApplication.CreateApplicationCriteria();
    $objApplicationCriteria.FileType =
[Centrify.DirectControl.API.WindowsFileType]::EXE;
    $objApplicationCriteria.FileName = "calc.exe";
    $objApplicationCriteria.Path = "SYSTEMPATH";
    $objApplicationCriteria.FileDescription = "Windows
Calculator";
    $objApplicationCriteria.FileDescriptionMatchOption =
[Centrify.DirectControl.API.StringMatchOption]::ExactMatch;
    $objApplicationCriteria.FileVersion = "6.1";
    $objApplicationCriteria.FileVersionMatchOption =
[Centrify.DirectControl.API.VersionMatchOption]::LaterThanOrEqualTo;
    $objApplicationCriteria.Description = "Match criteria for
Windows Calc";
$objWindowsApplication.ApplicationCriteriaList = $criteriaList;
    $objWindowsApplication.Commit();
    Write-Host("WindowsApplication {0} has been added to zone {1}
successfully." -f $strWindowsApplication, $strZone);
    exit 0;
}
```

## Priority

Gets or sets the priority of this right.

## Syntax

```
int Priority {get; set;}
```

## Property value

The priority of the right. Default is 0.



## Discussion

This number is used when handling multiple matches for rights specified by wild cards. If rights specified by this property object match rights specified by another property object, the object with the higher priority prevails. The higher the value of the `Priority` property, the higher the priority.

## Example

See [ApplicationCriteriaList](#) for a code sample that illustrates using `Priority` in a script.

## RequirePassword

Gets or sets whether the logged-in user's password is required when this right is used.

### Syntax

```
bool RequirePassword {get; set;}
```

### Property value

Set to true if the right requires the logged-in user's password.

## Example

See [ApplicationCriteriaList](#) for a code sample that illustrates using `RequirePassword` in a script.

## RunAs

Gets or sets the run-as property for this right.

### Syntax

```
string RunAs {get; set;}
```

### Property value

The run-as property for a single user.



## Discussion

If the `RunAsType` property is set to `Self`, the remote application is run under the logged-in user account, but with the additional privileges of the user whose SID is listed in the `RunAs` property. For example, if the `windowsApplication` right is set to run as `Self` and `RunAs` contains the SID of the Network Admins group, then this application runs with the permissions of the logged-in user plus the permissions of the Network Admins group.

If the `RunAsType` property is set to `User`, the remote application is run under the user whose SID is listed in the `RunAs` property. For example, if the `windowsApplication` right is set to run as `User` and `RunAs` contains the SID of the user `NetAdmin`, then this application runs with the permissions of the `NetAdmin` user.

If the `RunAs` property is empty, this right is invalid and an exception is thrown when you call the `Commit` method.

**Note:** This property is for use in VBScript programs. Use the `RunAsList` property for .NET.

## RunAsList

Gets or sets the run-as list for this right.

### Syntax

```
ICollection<SecurityIdentifier> RunAsList {get; set;}
```

### Property value

The run-as list for the right.

## Discussion

If the `RunAsType` property is set to `Self`, the application is run under the logged-in user account, but with the additional privileges of the groups whose SIDs are listed in the `RunAsList` property. For example, if the `windowsApplication` right is set to run as `Self` and `RunAsList` contains the SID of the Local Admins group, then this application runs with the permissions of the logged-in user plus the permissions of the Local Admins group.



If the `RunAsType` property is set to `User`, the application is run under the user whose SID is listed in the `RunAsList` property. In this case, the `RunAsList` property contains only a single SID. For example, if the `windowsApplication` right is set to run as `User` and `RunAsList` contains the SID of the user `Admin`, then this application runs with the permissions of the `Admin` user.

If the `RunAsList` property is empty, this right is invalid and an exception is thrown when you call the `Commit` method.

**Note:** This property can only be used in .NET programs. Use the `RunAs` property for VBScript.

## RunAsType

Gets or sets the run-as type for this right.

### Syntax

```
windowsRunAsType RunAsType {get; set;}
```

### Property value

The run-as type of the right.

Possible values:

```
public enum windowsRunAsType
{
    // Run as self
    Self,
    // Run as another user
    User
}
```

### Discussion

If the `RunAsType` property is set to `Self`, the application runs as the logged-in user with the additional privileges of the groups whose SIDs are listed in the `RunAsList` property. For example, if the `windowsApplication` right is set to run as `Self` and `RunAsList` contains the SID of the `Local Admins` group, then this application runs with the permissions of the logged-in user plus the permissions of the `Local Admins` group.

If the `RunAsType` property is set to `User`, the application is run as the user whose SID is listed in the `RunAsList` property. For example, if the `windowsApplication` right is set to run as `User` and `RunAsList` contains the



SID of the user Admin, then this application runs as Admin with the permissions of that user.

## Example

See [ApplicationCriteriaList](#) for a code sample that illustrates using `RunAsType` in a script.

## WindowsApplicationCriteria

This class represents the match criteria for identifying a Windows application right.

## Syntax

```
public interface IWindowsApplicationCriteria
```

## Methods

The `WindowsApplicationCriteria` class provides the following method.

This method	Does this
<code>validate</code>	Determines whether the match criteria is valid.

## Properties

The `WindowsApplicationCriteria` class provides the following match criteria properties that correspond to the fields displayed on the Match Criteria tab in Access Manager:

This property	Does this
<code>Argument</code>	Gets or sets the arguments allowed with this Windows application right.
<code>CompanyName</code>	Gets or sets the company name to match to identify the Windows application associated with this right.
<code>CompanyNameMatchOption</code>	Specifies whether the string specified for the <code>CompanyName</code> property must be an exact match or a



<b>This property</b>	<b>Does this</b>
	partial match.
<b>Description</b>	Gets or sets the description for the match criteria defined for a specific application right.
<b>FileDescription</b>	Gets or sets the file description to match to identify the Windows application associated with this right.
<b>FileDescriptionMatchOption</b>	Specifies whether the string specified for the FileDescription property must be an exact match or a partial match.
<b>FileHash</b>	Gets or sets the encrypted file hash to match to identify the Windows application associated with this right. The file hash is generated using the SHA-1 encryption algorithm, which is FIPS-compliant.
<b>FileName</b>	Gets or sets the file name to match to identify the Windows application associated with this right.
<b>FileType</b>	Gets or sets the type of executable file to match to identify the Windows application associated with this right.
<b>FileVersion</b>	Gets or sets the file version to match to identify the Windows application associated with this right.
<b>FileVersionMatchOption</b>	Specifies whether the version must be equal to, earlier than or equal to, or later than or equal to the version specified for the FileVersion property.
<b>IsArgumentCaseSensitive</b>	Specifies whether arguments for the Argument property are case-sensitive.
<b>IsArgumentExactMatch</b>	Specifies whether the string specified for the Argument property must be an exact match or a partial match.
<b>LocalOwner</b>	Gets or sets the local user name or group name to match to allow the use of this Windows application right.
<b>OwnerDN</b>	Specifies the distinguished name of the Active Directory user or group who is the file owner to match to identify the Windows application associated with this right.
<b>OwnerSid</b>	Specifies the security identifier of the Active Directory user or group who is the file owner to match to identify the Windows application associated with this right.
<b>OwnerType</b>	Gets or sets the type of owner to match to allow the use of this Windows application right.
<b>Path</b>	Gets or sets the path to the executable to match to identify the Windows application associated with this right.
<b>ProductName</b>	Gets or sets the product name to match to identify the Windows application associated with this right.
<b>ProductNameMatchOption</b>	Specifies whether the string specified for the ProductName property must be an exact match or a



This property	Does this
	partial match.
<b>ProductVersion</b>	Gets or sets the product version to match to identify the Windows application associated with this right.
<b>ProductVersionMatchOption</b>	Specifies whether the version must be equal to, earlier than or equal to, or later than or equal to the version specified for the ProductVersion property.
<b>Publisher</b>	Gets or sets the publisher name to match to identify the Windows application associated with this right.
<b>PublisherMatchOption</b>	Specifies whether the publisher must be an exact match, partial match, start with, or end with the string specified for the Publisher property.
<b>RequireAdministrator</b>	Specifies whether the Windows application associated with this right requires an administrative user account.
<b>SerialNumber</b>	Gets or sets the volume serial number to match to identify the Windows application associated with this right.
<b>SerialNumberMatchOption</b>	Specifies whether the volume serial number must be an exact match, partial match, start with, or end with the string specified for the SerialNumber property.

## Discussion

A Windows application right enables a user to run a Windows application with the privileges of another user or as a member of an Active Directory or built-in group. For example, you can use a Windows application right to give a standard Windows user elevated privileges to run a database management application as a database administrator. You can define the criteria to use to identify the Windows application associated with an application right using the `windowsApplicationCriteria` properties.

## Validate

Determines whether the match criteria is valid.

## Syntax

```
void validate()
```

## Discussion

This property is only applicable in .NET-compatible programs.



## Exception

Validate throws an ApplicationException if any property specified for the match criteria is not valid.

## Argument

Gets or sets the arguments allowed with this Windows application right.

If you specify a file type of .msc, you must also specify the Arguments property. The Arguments property is optional for all other file types. If this property is set to an empty string, no arguments are allowed. Do not specify the Arguments property if you are granting a right to access the application without argument restrictions. If the property is not defined, the application can be executed with any argument or combination of arguments.

## Syntax

```
string Argument {get; set;}
```

## Property value

A list of arguments that can be used with the application when this application is executed.

## CompanyName

Gets or sets the company name to match to identify the Windows application associated with this right.

## Syntax

```
string CompanyName {get; set;}
```

## Property value

The company name associated with the application.

## Example

The following code sample illustrates using CompanyName in a script:



```
...
$listType = ("System.Collections.Generic.List`1" -as
"Type");
$listType = $listType.MakeGenericType( @(
("Centrify.DirectControl.API.IWindowsApplicationCriteria" -as
"Type")));
$criteriaList = [Activator]::CreateInstance($listType);
$objApplicationCriteria =
$objWindowsApplication.CreateApplicationCriteria();
$objApplicationCriteria.FileType =
[Centrify.DirectControl.API.WindowsFileType]::EXE;
$objApplicationCriteria.FileName = "filename.exe";
$objApplicationCriteria.Path = "SYSTEMPATH";
$objApplicationCriteria.Argument = "Optional arguments 1";
$objApplicationCriteria.IsArgumentCaseSensitive = $true;
$objApplicationCriteria.IsArgumentExactMatch = $true;
$objApplicationCriteria.CompanyName = "Cendura Software";
$objApplicationCriteria.CompanyNameMatchOption =
[Centrify.DirectControl.API.StringMatchOption]::ExactMatch;
...
```

## CompanyNameMatchOption

Specifies whether the string specified for the `CompanyName` property must be an exact match or can be a partial match to identify an application associated with this right.

### Syntax

```
StringMatchOption CompanyNameMatchOption {get; set;}
```

### Property value

The match criteria operator, which specifies the type of matching to perform for the company name field.

Possible values:

```
public enum StringMatchOption
{
    // Exact match
    ExactMatch,
    // Partial match
    Contains
}
```

### Example

See [CompanyName](#) for code sample that illustrates using `CompanyNameMatchOption` in a script.



## Description

Gets or sets the description for the set of match criteria defined for a specific application right.

For example, if you are defining match criteria that will grant an application right for multiple versions of SQL Server Management Studio running on different versions of the Windows operating system, you might specify a description such as "SQL Server Management Studio (2005-2012)" to indicate the scope of the right.

## Syntax

```
string Description {get; set;}
```

## Property value

The descriptive text for a set of match criteria.

## Example

The following code sample illustrates using `Description` in a script:

```
$objWindowsApplication = $objZone.GetWindowsApplication  
($strWindowsApplication);  
{  
    $objWindowsApplication = $objZone.CreateWindowsApplication();  
    $objWindowsApplication.Name = $strWindowsApplication;  
    $objWindowsApplication.RunAsType = $runAsType;  
    $objWindowsApplication.RunAsString = $strDnList;  
    $objWindowsApplication.RequirePassword = $requirePassword;  
    $objWindowsApplication.Description = "SQL Server Match  
criteria";  
    $objWindowsApplication.Priority = 0;  
    ...  
}
```

## FileDescription

Gets or sets the file description to match to identify the Windows application associated with this right.

## Syntax

```
string FileDescription {get; set;}
```



## Property value

A file description for which to search.

## FileDescriptionMatchOption

Specifies whether the string specified for the FileDescription property must be an exact match or a partial match to identify an application associated with this right.

## Syntax

```
StringMatchOption FileDescriptionMatchOption {get; set;}
```

## Property value

The match criteria operator, which specifies the type of matching to perform for the file description field.

Possible values:

```
public enum StringMatchOption
{
    // Exact match
    ExactMatch,
    // Partial match
    Contains
}
```

## FileHash

Gets or sets the encrypted file hash to match to identify the Windows application associated with this right. The file hash is generated using the SHA-1 encryption algorithm, which is FIPS-compliant.

## Syntax

```
string FileHash {get; set;}
```

## Property value

The file hash to match to identify the application.



## FileName

Gets or sets the file name to match to identify the Windows application associated with this right.

### Syntax

```
string FileName {get; set;}
```

### Property value

The file name to match to identify the application.

## FileType

Gets or sets the type of executable file to match to identify the Windows application associated with this right. You must specify a file type to define a valid application right.

### Syntax

```
windowsFileType FileType {get; set;}
```

### Property value

The executable file type to match.

Possible values:

```
public enum windowsFileType
{
    // Batch file
    BAT,
    // Command script
    CMD,
    // Command file
    COM,
    // Control Panel Extension
    CPL,
    // Executable file
    EXE
    // Microsoft common console document
    MSC
    // windows installer package
    MSI
    // windows installer patch
    MSP
    // windows PowerShell cmdlet
```

• • • • •

```
    PS1
    // VBScript script
    VBS
    // windows script file
    WSF
}
```

## Example

The following code sample illustrates using `FileType` in a script:

```
$objWindowsApplication = $objZone.GetWindowsApplication
($strWindowsApplication);
{
    ...
    $listType = $listType.MakeGenericType( @
("Centrify.DirectControl.API.IWindowsApplicationCriteria" -as
"Type"));
    $criteriaList = [Activator]::CreateInstance($listType);
    $objApplicationCriteria =
$objWindowsApplication.CreateApplicationCriteria();
    $objApplicationCriteria.FileType =
[Centrify.DirectControl.API.WindowsFileType]::EXE;
    $objApplicationCriteria.FileName = "filename.exe";
    ...
}
```

## FileVersion

Gets or sets the file version to match to identify the Windows application associated with this right.

### Syntax

```
string FileVersion {get; set;}
```

### Property value

The file version to match to identify the application.

## FileVersionMatchOption

Specifies whether the version must be equal to, earlier than or equal to, or later than or equal to the version specified for the `FileVersion` property.



## Syntax

```
VersionMatchOption FileVersionMatchOption {get; set;}
```

## Property value

The match criteria operator, which specifies the type of matching to perform for the file version field.

Possible values:

```
public enum VersionMatchOption
{
    // Match the version specified
    Equal,
    // Match earlier than or equal to the specified version
    EarlierThanOrEqualTo,
    // Match later than or equal to the specified version
    LaterThanOrEqualTo
}
```

## IsArgumentCaseSensitive

Specifies whether the arguments for the Argument property are case-sensitive.

## Syntax

```
bool IsArgumentCaseSensitive {get; set;}
```

## Property value

Set to true if arguments are case-sensitive. The default is false.

## IsArgumentExactMatch

Specifies whether the arguments specified for the Argument property must be an exact match.

## Syntax

```
bool IsArgumentExactMatch {get; set;}
```

## Property value

Set to true if the arguments must be an exact match. The default is false.



## LocalOwner

Gets or sets the local owner to match to allow the use of this Windows application right.

### Syntax

```
string LocalOwner {get; set;}
```

### Property value

The local owner user name or group to match to execute the application.

## OwnerDN

Specifies the distinguished name of the Active Directory user or group who is the file owner to match to identify the Windows application associated with this right.

This property is only applicable in VBScript programs.

### Syntax

```
string OwnerDN {get; set;}
```

### Property value

The distinguished name of the owner of the file to match.

## OwnerSid

Specifies the security identifier of the Active Directory user or group who is the file owner to match to identify the Windows application associated with this right.

This property is only applicable in .NET-compatible programs.

### Syntax

```
SecurityIdentifier OwnerSid {get; set;}
```



## Property value

The security identifier of the owner of the file to match.

## OwnerType

Gets or sets the type of owner to match to allow the use of this Windows application right.

### Syntax

In .NET:

```
Nullable<WindowsFileOwnerType> OwnerType {get; set;}
```

In VBScript:

```
WindowsFileOwnerType OwnerType {get; set;}
```

## Property value

The type of owner to match.

Possible values:

```
public enum windowsFileOwnerType
{
    // Active Directory group or user SID
    Sid,
    // Local group account
    LocalGroup,
    // Local user account
    LocalUser
}
```

## Path

Gets or sets the path to the executable to match to identify the Windows application associated with this right.

### Syntax

```
string Path {get; set;}
```



## Property value

Path to the executable for the application. You can specify multiple paths separated by semi-colons (;). You can specify the standard system path using the “SYSTEMPATH” keyword, specify a full custom path, such as C:\windows\system32\inetsrv, or use one of the supported path variables.

The supported path variables are:

```
%systemroot%  
%system32%  
%syswow64%  
%program files%  
%program files(x86)%
```

The space between “program” and “files” is required.

## ProductName

Gets or sets the product name to match to identify the Windows application associated with this right.

### Syntax

```
string ProductName {get; set;}
```

## Property value

The product name to match to identify the application.

## ProductNameMatchOption

Specifies whether the string specified for the ProductName property must be an exact match or a partial match.

### Syntax

```
StringMatchOption ProductNameMatchOption {get; set;}
```

## Property value

The match criteria operator, which specifies the type of matching to perform for the product name field.



Possible values:

```
public enum StringMatchOption
{
    // Exact match
    ExactMatch,
    // Partial match
    Contains
}
```

## ProductVersion

Gets or sets the product version to match to identify the Windows application associated with this right.

### Syntax

```
string ProductVersion {get; set;}
```

### Property value

The product version to match to identify the application.

## ProductVersionMatchOption

Specifies whether the version must be equal to, earlier than or equal to, or later than or equal to the version specified for the ProductVersion property.

### Syntax

```
VersionMatchOption ProductVersionMatchOption {get; set;}
```

### Property value

The match criteria operator, which specifies the type of matching to perform for the product version field.

Possible values:

```
public enum VersionMatchOption
{
    // Match the version specified
    Equal,
    // Match earlier than or equal to the specified version
    EarlierThanOrEqualTo,
    // Match later than or equal to the specified version
}
```



```
        LaterThanOrEqualTo  
    }
```

## Publisher

Gets or sets the publisher information from a digital certificate to match to identify the Windows application associated with this right.

### Syntax

```
string Publisher {get; set;}
```

### Property value

The publisher information to match to identify the application.

## PublisherMatchOption

Specifies whether the publisher information must be an exact match, partial match, start with, or end with the string specified for the Publisher property.

### Syntax

```
StringMatchOption PublisherMatchOption {get; set;}
```

### Property value

The match criteria operator, which specifies the type of matching to perform for the publisher field.

Possible values:

```
public enum StringMatchOption  
{  
    // Exact match  
    ExactMatch,  
    // Partial match  
    Contains  
    // Starts with match  
    Startswith  
    // Ends with match  
    Endswith  
}
```



## RequireAdministrator

Specifies whether the Windows application associated with this right requires an administrative user account.

### Syntax

```
bool RequireAdministrator {get; set;}
```

### Property value

Set to `true` if the application must be executed using an administrative user account. The default is `false`.

## SerialNumber

Gets or sets the volume serial number to match to identify the Windows application associated with this right.

This property is used to match an application located on a CD or DVD media with the specified volume serial number. If the target application is not executed from CD or DVD media, this property does not apply and the application right will not be granted.

### Syntax

```
string SerialNumber {get; set;}
```

### Property value

The volume serial number information to match to identify the application.

## SerialNumberMatchOption

Specifies whether the volume serial number must be an exact match, partial match, start with, or end with the string specified for the `SerialNumber` property.

### Syntax

```
StringMatchOption SerialNumberMatchOption {get; set;}
```



## Property value

The match criteria operator, which specifies the type of matching to perform for the volume serial number field.

Possible values:

```
public enum StringMatchOption
{
    // Exact match
    ExactMatch,
    // Partial match
    Contains
    // Starts with match
    StartsWith
    // Ends with match
    EndsWith
}
```

## WindowsApplications

The windowsApplications class manages a collection of Windows application rights.

## Syntax

```
public interface IWindowsApplications
```

## Methods

The windowsApplications class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Gets the enumerator you can use to enumerate all windows application rights.

## GetEnumerator

Returns an enumeration of windowsApplication objects.



## Syntax

`IEnumerator GetEnumerator()`

## Return value

Returns an enumerator you can use to list all of the `WindowsApplication` objects.

# WindowsDesktop

This class represents a Windows desktop right.

## Syntax

```
public interface IWindowsDesktop:IRight
```

The `WindowsDesktop` class provides the following methods:

This method	Does this
<code>Commit</code>	Commits changes in the right to Active Directory. (Inherited from <code>Right</code> .)
<code>Delete</code>	Removes the right. (Inherited from <code>Right</code> .)

## Properties

The `WindowsDesktop` class provides the following properties:

This property	Does this
<code>Description</code>	Gets or sets the description of the right. (Inherited from <code>Right</code> .)
<code>IsReadable</code>	Indicates whether the right is readable. (Inherited from <code>Right</code> .)
<code>IsWritable</code>	Indicates whether the right is writable. (Inherited from <code>Right</code> .)
<code>Name</code>	Gets or sets the name of the right.



This property	Does this
	(Inherited from <a href="#">Right</a> .)
<a href="#">AddLocalGroupPartialProfile</a>	Gets or sets the priority of this right.
<a href="#">RequirePassword</a>	Gets or sets whether the user's password is required when this right is used.
<a href="#">RunAs</a>	Gets or sets the SID for the run-as user or an SID for the users assigned the right (VBScript).
<a href="#">RunAsList</a>	Gets or sets the SID for the run-as user or a list of SIDs for the users assigned the right (.NET).
<a href="#">RunAsType</a>	Gets or sets the run-as type for the right.
<a href="#">Zone</a>	Gets the zone this right belongs to. (Inherited from <a href="#">Right</a> .)

## Discussion

A Windows desktop right provides a complete desktop that behaves as if the user had logged in as specific privileged user with the privileges of another user or as a member of an Active Directory or built-in group. For example, you can use a Windows desktop right to give a standard Windows user elevated privileges to run local applications as a member of the built-in Administrators group.

## Priority

Gets or sets the priority of this right.

## Syntax

```
int Priority {get; set;}
```

## Property value

The priority of the right. Default is 0.

## Discussion

This number is used when handling multiple matches for rights specified by wild cards. If rights specified by this property object match rights specified by



another property object, the object with the higher priority prevails. The higher the value of the `Priority` property, the higher the priority.

## RequirePassword

Gets or sets whether the logged-in user's password is required when this right is used.

### Syntax

```
bool RequirePassword {get; set;}
```

### Property value

Set to true if the right requires the logged-in user's password.

## RunAs

Gets or sets the run-as property for this right.

### Syntax

```
string RunAs {get; set;}
```

### Property value

The run-as property for a single user.

### Discussion

If the `RunAsType` property is set to `self`, the remote application is run under the logged-in user account, but with the additional privileges of the groups whose SIDs are listed in the `RunAs` property as a semicolon (;) separated string. For example, if the `windowsDesktop` right is set to run as `self` and `RunAs` contains the SID of the Network Admins group, then this application runs with the permissions of the logged-in user plus the permissions of the Network Admins group.

If the `RunAsType` property is set to `User`, the remote application is run under the user whose SID is listed in the `RunAs` property. For example, if the `windowsDesktop` right is set to run as `User` and `RunAs` contains the SID of the



user NetAdmin, then this application runs with the permissions of the NetAdmin user.

If the RunAs property is empty, this right is invalid and an exception is thrown when you call the `Commit` method.

**Note:** This property is for use in VBScript programs. Use the `RunAsList` property for .NET.

## RunAsList

Gets or sets the run-as list for this right.

### Syntax

```
ICollection<SecurityIdentifier> RunAsList {get; set;}
```

### Property value

The run-as list for the right.

### Discussion

If the `RunAsType` property is set to `Self`, the desktop runs as the logged-in user with the additional privileges of the groups whose SIDs are listed in the `RunAsList` property. For example, if the `WindowsDesktop` right is set to run as `Self` and `RunAsList` contains the SID of the Local Admins group, then this desktop runs with the permissions of the logged-in user plus the permissions of the Local Admins group.

If the `RunAsType` property is set to `User`, the desktop is run as the user whose SID is listed in the `RunAsList` property. In this case, the `RunAsList` property contains only a single SID. For example, if the `WindowsDesktop` right is set to run as `User` and `RunAsList` contains the SID of the user Admin, then this desktop runs as Admin with the permissions of that user.

If the `RunAsList` property is empty, this right is invalid and an exception is thrown when you call the `Commit` method.

**Note:** This property can only be used in .NET programs. Use the `RunAs` property for VBScript.



## RunAsType

Gets or sets the run-as type for this right.

### Syntax

```
WindowsRunAsType RunAsType {get; set;}
```

### Property value

The run-as type of the right.

Possible values:

```
public enum WindowsRunAsType
{
    // Run as self
    Self,
    // Run as another user
    User
}
```

### Discussion

If the `RunAsType` property is set to `Self`, the desktop runs as the logged-in user with the additional privileges of the groups whose SIDs are listed in the `RunAsList` property. For example, if the `WindowsDesktop` right is set to run as `Self` and `RunAsList` contains the SID of the Local Admins group, then this desktop runs with the permissions of the logged-in user plus the permissions of the Local Admins group.

If the `RunAsType` property is set to `User`, the desktop is run as the user whose SID is listed in the `RunAsList` property. For example, if the `WindowsDesktop` right is set to run as `User` and `RunAsList` contains the SID of the user Admin, then this desktop runs as Admin with the permissions of that user.

## WindowsDesktops

The `WindowsDesktops` class manages a collection of Windows desktop rights.

### Syntax

```
public interface IWindowsDesktops
```



## Methods

The windowsDesktops class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Gets the enumerator you can use to enumerate all windows desktop rights.

## GetEnumerator

Returns an enumeration of windowsDesktop objects.

## Syntax

```
IEnumerator GetEnumerator()
```

## Return value

Returns an enumerator you can use to list all the windowsDesktop objects.

## WindowsUser

The windowsUser class manages the Windows user profile information of a user.

## Syntax

```
public interface IWindowsUser
```

## Methods

The windowsUser class provides the following methods:

This method	Does this
<code>AddUserRoleAssignment</code>	Adds a role assignment to the user profile.
<code>GetDirectoryEntry</code>	Returns the directory entry for the user from Active Directory.



This method	Does this
<code>GetEffectiveUserRoleAssignments</code>	Returns an enumeration of the effective user role assignments.

## Properties

The `WindowsUser` class provides the following property:

This property	Does this
<code>Name</code>	Gets the Windows login name of the user.

## Discussion

A user's Windows profile consists of the information used by Windows to identify the user. See the `HierarchicalUser` class for a user's UNIX profile.

The rights you assign to users and group in a particular role apply to Active Directory users and groups. They can also apply to locally-defined users and groups if you configure the role definition to allow local accounts to be assigned to the role. All Windows users, including local users, must be assigned at least one role that allows them log on locally, remotely, or both.

## AddUserRoleAssignment

Adds a role assignment to the user profile.

### Syntax

```
IRoleAssignment AddUserRoleAssignment(IHierarchicalZone zone)  
IRoleAssignment AddUserRoleAssignment(IHierarchicalZoneComputer  
computer)
```

### Parameters

Specify one of the following parameters when using this method.

Parameter	Description
<code>zone</code>	The zone to which the Windows user belongs.
<code>computer</code>	The computer to which the Windows user belongs.



## Return value

An empty user role assignment object. This role assignment is not stored in Active Directory until you call the `RoleAssignment.Commit` method.

## Discussion

When you assign computer-level overrides for user, group, or computer role assignments, internally Centrify creates a computer zone object, which is a special type of zone that contains the users, groups, and computer role assignments that are specific to only that one computer. Use the second form of this method to obtain a computer-level role assignment for this user.

## Exceptions

`AddUserRoleAssignment` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `ApplicationException` if the parameter value is not a valid user or if the method failed to create a role assignment because it cannot find the user.

## GetDirectoryEntry

Returns the directory entry for the user from Active Directory.

## Syntax

```
DirectoryEntry GetDirectoryEntry()
```

## Return value

A directory entry for the service connection point that represents the user's Windows profile.

## Discussion

The `DirectoryEntry` object represents the directory object for the user and its associated attributes.

**Note:** This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.



## Exceptions

`GetDirectoryEntry` throws an `ApplicationException` if it cannot get the directory object.

## GetEffectiveUserRoleAssignments

Returns an enumeration of the effective user role assignments.

### Syntax

```
IRoleAssignments GetEffectiveUserRoleAssignments()
```

### Return value

An enumeration of the effective user role assignments for this user.

### Discussion

The collection of effective role assignments is a combination of all the role assignments for this user in this zone and all parent zones. See the [HierarchicalUser](#) class for a more complete discussion.

## Name

Gets the Windows login name of the user.

### Syntax

```
string Name {get;}
```

### Property value

The login name from the user's Windows profile.

## WindowsUsers

The `windowsUsers` class manages a collection of Windows user objects.

• • • • •

## Syntax

```
public interface IWindowsUsers
```

## Methods

The windowsUsers class provides the following method:

This method	Does this
<code>GetEnumerator</code>	Gets the enumerator you can use to enumerate all Windows user objects.

## GetEnumerator

Returns an enumeration of windowsUser objects.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

Returns an enumerator you can use to list all the windowsUser objects.

## Zone

Manages Centrifly zone objects (`Centrifly.DirectControl.API.IZone`).

## Syntax

```
public interface IZone
```

## Discussion

For each zone you create, you must also define several zone properties. You can also use the Zone class to manage user access rights and the actions users are allowed to perform within a zone. For more information about creating and



working with Centrify zones interactively using the Access Manager console, see the *Administrator's Guide for Linux and UNIX*.

## Methods

The Zone class provides the following methods:

This method	Does this
<code>AddMitUser</code>	Adds an MIT Kerberos realm-trusted user to this zone.
<code>Commit</code>	Commits settings to Active Directory for the zone object.
<code>CreateImportPendingGroup</code>	Creates a "pending import" group in the zone.
<code>CreateImportPendingUser</code>	Creates a "pending import" user in the zone.
<code>Delete</code>	Deletes the zone object from Active Directory.
<code>GetComputerByDN</code>	Returns the computer profile using the distinguished name (DN) of the profile.
<code>GetComputers</code>	Returns the list of computers in the zone.
<code>GetComputersContainer</code>	Returns the directory entry for the <code>Computers</code> parent container object.
<code>GetDirectoryEntry</code>	Returns the directory entry for the zone.
<code>GetDisplayName</code>	Returns the display name of the zone.
<code>GetGroupsContainer</code>	Returns the directory entry for the Groups parent container object.
<code>GetGroupUnixProfile</code>	Returns the UNIX group profile for a specified group in the zone.
<code>GetGroupUnixProfileByDN</code>	Returns the group profile using the distinguished name (DN) of the profile.
<code>GetGroupUnixProfileByName</code>	Returns the UNIX group profile for a specified group name in the zone.
<code>GetGroupUnixProfiles</code>	Returns the list of UNIX groups in the zone.
<code>GetImportPendingGroup</code>	Returns an individual "pending import" group in the zone.
<code>GetImportPendingGroups</code>	Returns the collection of "pending import" groups in the zone.
<code>GetImportPendingUser</code>	Returns an individual "pending import" user in the zone.
<code>GetImportPendingUsers</code>	Returns the collection of "pending import" users in the zone.
<code>GetLocalGroupsContainer</code>	Returns the DirectoryEntry of the local groups container.



This method	Does this
<code>GetLocalGroupUnixProfile</code>	Returns the local UNIX group profile for a specified group name in the zone.
<code>GetLocalGroupUnixProfileByDN</code>	Returns a local group profile using the distinguished name (DN) of the profile.
<code>GetLocalGroupUnixProfileByGid (Int32)</code>	Returns the local group profile using the Group Identifier (GID). This method is exposed to the .COM interface.
<code>GetLocalGroupUnixProfiles</code>	Returns a list of the local group profiles in the zone.
<code>GetLocalUsersContainer</code>	Returns the directory entry of the local users container.
<code>GetLocalUserUnixProfile</code>	Returns the local user profile using the specified user name.
<code>GetLocalUserUnixProfileByDN</code>	Returns the local user profile specified by the distinguished name (DN) of the profile.
<code>GetLocalUserUnixProfileByUid (Int32)</code>	Returns the local user profile using the User Identifier (UID). This method is exposed to the .COM interface
<code>GetLocalUserUnixProfiles</code>	Returns a list of the local user profiles in the zone.
<code>GetUsersContainer</code>	Returns the directory entry for the Users parent container object.
<code>GetUserUnixProfileByDN</code>	Returns the user profile using the distinguished name (DN) of the profile.
<code>GetUserUnixProfileByName</code>	Returns the UNIX user profile for a specified user name in the zone.
<code>GetUserUnixProfiles</code>	Returns the list of UNIX users in the zone.
<code>GroupUnixProfileExists</code>	Indicates whether a UNIX profile exists for the specified group in the zone.
<code>LocalGroupUnixProfileExists</code>	Indicates whether a UNIX profile exists in the zone for the specified local group.
<code>LocalUserUnixProfileExists</code>	Indicates whether a UNIX profile exists in the zone for the specified local user.
<code>PrecreateComputer</code>	Adds a computer to the zone.
<code>PrecreateWindowsComputer</code>	Adds a Windows computer to the zone.
<code>Refresh</code>	Returns the data stored for the zone object from the data in the Active Directory entry.
<code>UserUnixProfileExists</code>	Indicates whether a UNIX profile exists for the specified user in the zone.

## Properties

The Zone class provides the following properties:



<b>This property</b>	<b>Does this</b>
<b>AdsiInterface</b>	Gets the IADs interface of the zone object in Active Directory.
<b>ADsPath</b>	Gets the LDAP path to the zone object.
<b>AgentlessAttribute</b>	Gets or sets the Active Directory attribute used for storing the user's password hash.
<b>AvailableShells</b>	Gets or sets the list of available shells for the zone.
<b>Cims</b>	Gets the Cims object managing the zone.
<b>DefaultGroup</b>	Gets or sets the default group profile to use as the primary group for new users in the zone.
<b>DefaultHomeDirectory</b>	Gets or sets the default path to the user's home directory for new users in the zone.
<b>DefaultShell</b>	Gets or sets the default shell assigned to new users in the zone.
<b>DefaultValueZone</b>	Gets or sets the zone to use for default zone values.
<b>Description</b>	Gets or sets the description property for the zone.
<b>FullName</b>	Gets the full name of the zone.
<b>GroupAutoProvisioningEnabled</b>	Indicates whether auto-provisioning of group profiles is enabled for the zone.
<b>ID</b>	Gets the unique identifier for the zone.
<b>IsHierarchical</b>	Indicates whether this zone supports hierarchical zone features.
<b>IsReadable</b>	Indicates whether the zone object's properties are readable.
<b>IsSFU</b>	Indicates whether the zone uses the Microsoft Services for UNIX (SFU) schema extension.
<b>IsTruncateName</b>	Determines whether the zone is a TruncateName zone.
<b>IsWritable</b>	Indicates whether the zone object's properties are writable.
<b>Licenses</b>	Gets or sets the license container associated with this zone.
<b>MasterDomainController</b>	Gets or sets the name of the primary domain controller for the zone.
<b>MustMaintainADGroupMembership</b>	Determines whether Active Directory group membership must be maintained for UNIX users in the zone.
<b>Name</b>	Gets or sets the name of the zone.
<b>NextAvailableGID</b>	Gets or sets the next available GID value for new groups in the zone.
<b>NextAvailableUID</b>	Gets or sets the next available UID value for new users



This property	Does this
	in the zone.
<b>NextGID</b>	Gets or sets the next GID to be used when adding users.
<b>NextUID</b>	Gets or sets the next UID to be used when adding users.
<b>NISDomain</b>	Gets or sets the NIS domain associated with the zone for SFU zones.
<b>ReservedGID</b>	Gets or sets the list of group identifiers (GIDs) that cannot be assigned in the zone.
<b>ReservedUID</b>	Gets or sets the list of User identifiers (UIDs) that cannot be assigned in the zone.
<b>Schema</b>	Gets the schema type of the zone object.
<b>SFUDomain</b>	Gets or sets the Active Directory domain associated with the zone for SFU zones.
<b>UserAutoProvisioningEnabled</b>	Indicates whether auto-provisioning of user profiles is enabled for the zone.
<b>Version</b>	Gets the version number of the data schema.

## AddMitUser

Adds a UNIX profile for a user in a trusted Kerberos realm to the zone.

### Syntax

```
IUserUnixProfile AddMitUser(string fullMitUserName, int uid, string name, string shell, string homeDir, int primaryGroup)
```

```
IUserUnixProfile AddMitUser(string fullMitUserName, long uid, string name, string shell, string homeDir, long primaryGroup)
```

### Parameters

Specify the following parameters when using this method.

Parameter	Description
<code>fullMitUserName</code>	The full user name of the user and the trusted Kerberos realm. For example: <code>username@mit.realm.name</code>
<code>uid</code>	The value to use as the UID of the user in the specified zone.
<code>name</code>	The UNIX profile name of the user in the specified zone.



Parameter	Description
shell	The default shell for the user in the specified zone.
homeDir	The default login shell for the user in the specified zone.
primaryGroup	The value to use as the GID for the user's primary group.

## Return value

A new `UserUnixProfile` object.

## Discussion

This method creates a UNIX profile object for a user in a trusted realm that is not tied to an Active Directory user object.

There are two versions of this method: one designed for COM-based programs that supports a 32-bit signed number for the `uid` and `primaryGroup` arguments and one designed for .NET-based programs that allows a 64-bit signed number for the arguments.

## Exceptions

`AddMitUser` may throw one of the following exceptions:

- `ApplicationException` if you try to add a Kerberos user to an SFU zone.
- `NotSupportedException` if the computer zone schema is not supported.

## Example

The following code sample illustrates using `AddMitUser` in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZoneByPath("LDAP://CN=cohesion_div,  
CN=zones,CN=centrify,CN=program data,DC=arcade,DC=com")  
'Create the UNIX profile for the Kerberos user "lewis.cain"  
set objUserUnixProfile = objUser.AddMitUser  
("lewis.cain@cohesion.org",98566,"cain1", "/bin/csh",  
"home/cain1", 98556)  
'Enable the UNIX profile for the new user and update AD  
objUserUnixProfile.UnixEnabled = True  
objUserUnixProfile.commit  
...
```



## Commit

Commits the settings or changes for a zone object to Active Directory.

### Syntax

```
void Commit()
```

### Discussion

This method confirms that the zone name and `DirectoryEntry` attributes are specified before updating Active Directory.

By default, the user who creates the zone object in Active Directory has permission to perform all administrative tasks in the zone. For information about setting and modifying the permissions required to perform specific tasks, see the *Planning and Deployment Guide*.

### Exceptions

`Commit` may throw one of the following exceptions:

- `ArgumentException` if the zone name is not valid.
- `ApplicationException` if the container is not a valid zone container, the zone name is already in use, or you have insufficient access rights to modify the zone.
- `UnauthorizedAccessException` if you have insufficient access rights to commit the data object.
- `COMException` if an LDAP error occurred. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

### Example

The following code sample illustrates using this method in a script:

```
...  
'Create a new zone.  
set objZone = Cims3.CreateZone(objContainer, strZone)  
'set the starting UID and GID for the new zone.  
objZone.nextAvailableUID = 10000  
objZone.nextAvailableGID = 10000  
'set the default shell and home directory the new zone.  
objZone.DefaultShell = "/bin/bash"
```

• • • • •

```
objZone.DefaultHomeDirectory = "/home/${user}"  
'Finalize the transaction and update Active Directory.  
objZone.Commit  
...
```

## CreateImportPendingGroup

Creates a “pending import” group in the zone.

### Syntax

```
IGroupInfo CreateImportPendingGroup (string source, DateTime  
timestamp)
```

### Parameters

Specify the following parameters when using this method.

Parameter	Data type	Description
source	String	The location of the source data for the group to be imported.
timestamp	DateTime	The date and time at which the data was retrieved.

### Return value

The newly created pending import group object.

### Discussion

Group profiles in a “pending import” group object needed to be mapped to Active Directory groups before they can be used. Groups in this state are normally imported from NIS domains or from text files and stored temporarily either in Active Directory or XML files until they are mapped to Active Directory accounts. For more information about importing and mapping groups, see the *Administrator’s Guide for Linux and UNIX*.

### Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Create the Pending Import group object  
set objPendGrp = objZone.CreateImportPendingGroup("script file",  
now)  
objPendGrp.Gid = 500
```

• • • • •

```
objPendGrp.Name = "users"  
objPendGrp.Commit  
...
```

## CreateImportPendingUser

Creates a “pending import” user in the zone.

### Syntax

```
IUserInfo CreateImportPendingUser(string source, DateTime  
timestamp)
```

### Parameters

Specify the following parameters when using this method.

Parameter	Description
source	The location of the source data for the user to be imported.
timestamp	The date and time at which the data was retrieved.

### Return value

The newly created pending import user object.

### Discussion

User profiles in a pending import user object need to be mapped to Active Directory groups before they can be used. Users in this state are normally imported from NIS domains or from text files and stored temporarily either in Active Directory or in XML files until they are mapped to Active Directory accounts. For more information about importing and mapping users, see the *Administrator's Guide for Linux and UNIX*.

### Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Create the Pending Import User object  
set objPendUsr = objZone.CreateImportPendingUser("script file",  
now)  
objPendUsr.Uid = 500  
objPendUsr.Name = "joe"
```

• • • • •

```
objPendUsr.PrimaryGroupId = 500
objPendUsr.HomeDirectory = "/home/joe"
objPendUsr.Shell = "/bin/bash"
objPendUsr.Gecos = "Joe Jane"
objPendUsr.Commit
...
```

## Delete

Deletes the zone object from Active Directory.

## Syntax

```
void Delete()
```

## Discussion

To delete a zone, you must have the `DeleteSubTree` privilege on the zone's parent container. For example, to delete a zone from the default location for new zones, you must have the right to `DeleteSubTree` allowed on the `domain/Program Data/Centrify/Zones` container object.

## Exceptions

`Delete` may throw one of the following exceptions:

- `UnauthorizedAccessException` if you have insufficient access rights to delete the zone.
- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using this method in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")
'Delete the zone object
objZone.Delete
...
```

• • • • •

## GetComputerByDN

Returns the computer profile for a computer in the zone using the distinguished name (DN) of the profile.

### Syntax

```
IComputer GetComputerByDN(string dn)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
dn	The distinguished name (DN) of the computer profile.

### Return value

The computer profile with the distinguished name (DN) matching the distinguished name specified, or null if no matching computer profile is found.

### Discussion

The computer profile is the service connection point associated with the computer object.

### Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZoneByPath  
("LDAP://CN=default,CN=zones,CN=centrifly,CN=program  
data,DC=arcade,DC=com")  
'Get the computer profile by DN  
set objComputer= objZone.GetComputerByDN  
("CN=velvet,CN=Computers,CN=default,CN=Zones,  
CN=centrifly,CN=program data,DC=arcade,DC=com")  
wscript.Echo computer.Name  
...
```

## GetComputers

Returns the list of computers in the zone.

• • • • •

## Syntax

```
IComputers GetComputers()
```

## Return value

The list of computers for the selected zone object.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZoneByPath  
("LDAP://CN=research,CN=zones,CN=centrify,CN=program data,DC=arca  
de,  
DC=com")  
'Display the list of computers in the zone  
wScript.Echo "Computers in zone: "  
for each computer in objZone.GetComputers  
    wScript.Echo computer.Name  
next  
...
```

## GetComputersContainer

Returns the parent container object for computer profiles in the zone.

## Syntax

```
DirectoryEntry GetComputersContainer()
```

## Return value

The `DirectoryEntry` object of the zone's Computers container.

## Discussion

If the Computers container does not exist, this method creates one for you.

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.



## Exceptions

GetComputersContainer may throw the following exception:

- ApplicationException if you try to use this method in a COM-based program.

## GetDirectoryEntry

Returns an instance of the DirectoryEntry object for the zone.

### Syntax

```
DirectoryEntry GetDirectoryEntry()
```

### Return value

The DirectoryEntry of the zone.

### Discussion

This method can only be used in .NET programs because DirectoryEntry is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

## GetDisplayName

Returns the name displayed for the zone in Access Manager.

### Syntax

```
string GetDisplayName()
```

### Return value

The DisplayName property for the selected zone object. For example, if using OU=UNIX,OU=Zones for the zone named qa:

```
domain/UNIX/Zones/qa
```

If the zone is defined as a Services for UNIX (SFU) zone, the DisplayName returned ends in SFU. For example:

```
domain/UNIX/Zones/qaSFU
```



## Discussion

In most cases, this method returns the same value as the zone's `FullName` property, for example, `domain/UNIX/Zones/testing_lab`, and they can be used interchangeably. However, if the zone is defined as a Services for UNIX (SFU) zone supporting the Microsoft Services for UNIX (SFU) schema, this method appends `[SFU]` to the zone name, for example `domain/UNIX/Zones/testing_lab [SFU]`.

## Example

The following code sample illustrates using this method in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath
("LDAP://CN=research,CN=zones,CN=centrify,CN=program data,DC=arcae,
DC=com")
'Display the name of the zone
wScript.Echo "Zone name: " & objZone.GetDisplayName
...
```

## GetGroupsContainer

Returns the directory entry for the parent container object for the group profiles in the zone.

### Syntax

```
DirectoryEntry GetGroupsContainer()
```

### Return value

The directory entry of the zone's Groups container.

## Discussion

If the Groups container does not exist, this method creates one for you.

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.



## Exceptions

`GetGroupsContainer` throws an `ApplicationException` if you try to use this method in a COM-based program.

## GetGroupUnixProfile

Returns the UNIX group profile for a specified group in the zone.

### Syntax

```
IGroupUnixProfile GetGroupUnixProfile(IGroup group)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>group</code>	The group for which you want to retrieve profile information.

### Return value

The `GroupUnixProfile` object for the specified group in the zone.

### Discussion

This method uses the `Centrify.DirectControl.API.IGroup` group returned by a `Cims.GetGroup` or `Cims.GetGroupByPath` call to retrieve the group profile.

## Exceptions

`GetGroupUnixProfile` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `NotSupportedException` if the zone schema is not supported.
- `ApplicationException` if there is more than one instance of the specified group in the zone.

## Example

The following code sample illustrates using this method in a script:



```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Identify the Active Directory group object  
set objGrp = cims.GetGroupByPath("LDAP://CN=berlin_  
qa,CN=Users,DC=ajax,DC=org")  
set objGrpProfile = objZone.GetGroupUnixProfile(objGrp)  
'Display the UNIX profile name for the group "berlin_qa"  
wScript.Echo objGrpProfile.Name  
...
```

## GetGroupUnixProfileByDN

Returns the UNIX profile for a group in the zone using the distinguished name (DN) of the profile.

### Syntax

```
IGroupUnixProfile GetGroupUnixProfileByDN(string dn)
```

### Parameter

Specify the following parameters when using this method.

Parameter	Description
dn	The distinguished name (DN) of the group profile.

### Return value

The group profile with the distinguished name (DN) matching the distinguished name specified, or null if no matching group profile is found.

### Discussion

The group profile is the service connection point associated with the Active Directory group object.

### Exceptions

GetGroupUnixProfile throws a `NotSupportedException` if the zone schema is not supported.

### Example

The following code sample illustrates using this method in a script:



```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZoneByPath  
("LDAP://CN=default,CN=zones,CN=centrify,CN=program data,DC=arcad  
e,DC=com")  
'Get the group profile by DN  
set objComputer= objZone.GetGroupUnixProfileByDN  
("CN=legal,CN=Groups,CN=default, CN=Zones,CN=centrify,CN=program  
data,DC=arcade,DC=com")  
...
```

## GetGroupUnixProfileByName

Returns the UNIX group profile for a group with the specified name in the zone.

### Syntax

```
IGroupUnixProfile GetGroupUnixProfileByName(string name)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The name of the UNIX group profile for which you want to retrieve information.

### Return value

The `GroupUnixProfile` object for the specified group name in the selected zone, or `null` if no group unix profile is found.

### Discussion

The name you specify should be the UNIX group name for the group if it differs from the Active Directory name for the group.

### Exceptions

`GetGroupUnixProfileByName` may throw one of the following exceptions:

- `NotSupportedException` if the zone schema is not supported.
- `ApplicationException` if there is more than one instance of the specified group in the zone.

• • • • •

## Example

The following code sample illustrates using this method in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")
'Get the UNIX profile for the group "berlin_qa"
set objGrp = objZone.GetGroupUnixProfileByName("berlin_qa")
'Display the GID for the group "berlin_qa"
wScript.Echo "GID: " & objGrp.GID
...
```

## GetGroupUnixProfiles

Returns the list of UNIX group profiles that have been defined for the zone.

### Syntax

```
IGroupUnixProfiles GetGroupUnixProfiles()
```

### Return value

The GroupUnixProfiles object for the zone.

## Example

The following code sample illustrates using this method in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath
("LDAP://CN=research,CN=zones,CN=centrify,CN=program data,DC=arcae,
DC=com")
'Display the list of group profiles defined for the zone
set objGroupProfiles = objZone.GetGroupUnixProfiles
for each objProfile in objGroupProfiles
    wScript.Echo "Group profile name: " & objProfile.Name
next
...
```

## GetImportPendingGroup

Returns an individual "pending import" group with the specified ID in the zone.



## Syntax

`IGroupInfo GetImportPendingGroup(string id)`

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>id</code>	The GUID of the “pending import” group profile for which you want to retrieve information.

## Return value

The `IGroupInfo` object for the specified ID in the zone.

## Discussion

Group profiles that are “pending import” are normally imported from NIS domains or from text files and not yet mapped to Active Directory groups. For more information about importing and mapping groups, see the *Administrator’s Guide for Linux and UNIX*.

## Example

The following code sample illustrates using this method in a script:

```
...
'Need to obtain an active directory container object
'Configure the test container.
Set objRootDSE = GetObject("LDAP://rootDSE")
set objContainer = GetObject("LDAP://" & strParent & "," &
objRootDSE.Get("defaultNamingContext"))
strContainerDN = objContainer.get("DistinguishedName")
'Get the zone object.
Set objZone = cims.GetZoneByPath("cn=" & strZone & "," &
strContainerDN)
Set objGroupInfo = objZone.GetImportPendingGroup(strID)
objGroupInfo.Delete
...
```

## GetImportPendingGroups

Returns the list of “pending import” groups for the zone.

• • • • •

## Syntax

IGroupInfos GetImportPendingGroups()

## Return value

The collection of “pending import” group profiles for the zone.

## Example

The following code sample illustrates using this method in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath
("LDAP://CN=default,CN=zones,CN=centrify,CN=program data,
DC=arcade,DC=com")
'Get the collection of pending import groups
set objPendingGrps = objZone.GetImportPendingGroups
if not objPendingGrps is nothing then
    wScript.Echo "Pending import groups: ", objPendingGrps.Count
    for each objPendingGrp in objPendingGrps
        wScript.Echo objPendingGrp.Gid, objPendingGrp.Name
        if objPendingGrp.Source = "script file" then
            objPendingGrp.Delete
        end if
    next
    wScript.Echo ""
end if
...
```

## GetImportPendingUser

Returns an individual “pending import” user with the specified ID in the zone.

## Syntax

IUserInfo GetImportPendingUser(string id)

## Parameter

Specify the following parameter when using this method:

Parameter	Description
id	The GUID of the “pending import” user profile for which you want to retrieve information.

• • • • •

## Return value

The `UserInfo` object for the specified ID in the zone.

## Discussion

User profiles that are “pending import” are normally imported from NIS domains or from text files and not yet mapped to Active Directory users. For more information about importing and mapping groups, see the *Administrator’s Guide for Linux and UNIX*.

## Example

The following code sample illustrates using this method in a script:

```
...
// Get the user object
IUser objUser = cims.GetUserByPath(strUser);
// Get the zone object
IZone objZone = cims.GetZoneByPath("cn=" + strZone + "," +
strContainerDN);
Import.IUserInfo objUserInfo = objZone.GetImportPendingUser
(strUserInfo);
if (objUser == null)
{
    Console.WriteLine("User " + strUser + " does not exist.");
}
else
{
    objUserInfo.Import(objUser);
}
...
```

## GetImportPendingUsers

Returns the list of “pending import” users for the zone.

### Syntax

```
IUserInfos GetImportPendingUsers()
```

## Return value

The collection of “pending import” user profiles for the zone.

## Example

The following code sample illustrates using this method in a script:

• • • • •

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Get the collection of pending users  
set objPendingUsrs = objZone.GetImportPendingUsers  
if not objPendingUsrs is nothing then  
    wScript.Echo "Pending import users: ", objPendingUsrs.Count  
    for each objPendingUsr in objPendingUsrs  
        wScript.Echo objPendingUsr.Uid, objPendingUsr.Name  
        if objPendingUsr.Source = "script file" then  
            objPendingUsr.Delete  
        end if  
    next  
    wScript.Echo ""  
end if  
...
```

## GetLocalGroupsContainer

Returns the directory entry for the parent container object for the local group profiles.

### Syntax

```
DirectoryEntry GetLocalGroupsContainer()
```

### Return value

The directory entry of the local group's container.

### Discussion

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

### Exceptions

`GetLocalGroupsContainer` may throw the following exception:

`ApplicationException` if you try to use this method in a COM-based program.

## GetLocalGroupUnixProfile

Returns the UNIX group profile for a specified local group.



## Syntax

```
IGroupUnixProfile GetLocalGroupUnixProfile(string groupName)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
groupName	The name of the local group for which you want to retrieve profile information.

## Return value

The `GroupUnixProfile` object for the specified local group name. If there is no group, `null` is returned.

## Exceptions

`GetGroupUnixProfile` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.

## GetLocalGroupUnixProfileByDN

Returns the Local UNIX profile for a group in the zone using the distinguished name (DN) of the profile.

## Syntax

```
IGroupUnixProfile GetLocalGroupUnixProfileByDN(string dn)
```

## Parameter

Specify the following parameters when using this method.

Parameter	Description
dn	The distinguished name (DN) of the local group profile.

## Return value

The local group profile with the distinguished name (DN) matching the distinguished name specified, or `null` if no matching group profile is found.



## GetLocalGroupUnixProfileByGid (Int32)

Returns the Local UNIX profile for a group in the zone using the group identifier (GID) of the profile. This method is exposed to the .COM interface.

### Syntax

```
IGroupUnixProfile GetLocalGroupUnixProfileByGid(int gid)
```

### Parameter

Specify the following parameters when using this method.

Parameter	Description
gid	The group identifier (GID) of the local group profile.

### Return value

The local group profile with the specified group identifier (GID) or null if no matching group profile is found.

## GetLocalGroupUnixProfiles

Get the list of local group profiles in the zone.

### Syntax

```
IGroupUnixProfiles GetLocalGroupUnixProfiles()
```

### Return value

Returns a collection of GroupUnixProfile objects. If there are no groups, null is returned.

## GetLocalUsersContainer

Returns the directory entry for the parent container object for the local user profiles in the zone.



## Syntax

```
DirectoryEntry GetLocalUsersContainer()
```

## Return value

The directory entry of the zone's users container.

## Discussion

If the Users container does not exist, this method creates one for you.

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

## Exceptions

`GetUsersContainer` may throw the following exception:

- `ApplicationException` if you try to use this method in a COM-based program.

## GetLocalUserUnixProfile

Returns the UNIX user profile for a specified local group.

## Syntax

```
IUserUnixProfile GetLocalUserUnixProfile(string userName)
```

## Parameter

Specify the `userName` parameter when using this method.

## Return value

Returns the local user profile with the specified user name. If there is no group, `null` is returned.



## GetLocalUserUnixProfileByDN

Returns the local UNIX profile for a user in the zone using the distinguished name (DN) of the profile.

### Syntax

```
IUserUnixProfile GetLocalUserUnixProfileByDN(string dn)
```

### Parameter

Specify the following parameters when using this method.

Parameter	Description
dn	The distinguished name (DN) of the local user profile.

### Return value

Returns the local user profile with the distinguished name (DN) matching the distinguished name specified, or null if no matching group profile is found.

## GetLocalUserUnixProfileByUid (Int32)

Returns the local UNIX profile for a user in the zone using the user identifier (UID) of the profile. This method is exposed to the .COM interface.

### Syntax

```
IUserUnixProfile GetLocalUserUnixProfileByUid(int uid)
```

### Parameter

Specify the following parameters when using this method.

Parameter	Description
uid	The user identifier (UID) of the local user profile.

### Return value

The local user profile with the specified user identifier (UID) or null if no matching user profile is found.



## GetLocalUserUnixProfiles

Get a list of local UNIX user profiles in the zone.

### Syntax

```
IUserUnixProfiles GetLocalUserUnixProfiles()
```

### Return value

Returns a collection of local user profiles in the zone. If there are no users, null is returned.

## GetUsersContainer

Returns the directory entry for the parent container object for the user profiles in the zone.

### Syntax

```
DirectoryEntry GetUsersContainer()
```

### Return value

The directory entry of the zone's Users container.

### Discussion

If the Users container does not exist, this method creates one for you.

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

### Exceptions

`GetUsersContainer` may throw the following exception:

- `ApplicationException` if you try to use this method in a COM-based program.



## GetUserUnixProfileByDN

Returns the UNIX profile for a user in the zone using the distinguished name (DN) of the profile.

### Syntax

```
IUserUnixProfile GetUserUnixProfileByDN(string dn)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
dn	The distinguished name (DN) of the user profile.

### Return value

The user profile with the distinguished name (DN) matching the distinguished name specified, or null if no matching user profile is found.

### Discussion

The user profile is the service connection point associated with the Active Directory user object.

### Exceptions

GetUserUnixProfileByDN throws a `NotSupportedException` if the zone schema is not supported.

### Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZoneByPath  
("LDAP://CN=default,CN=zones,CN=centrify,CN=program data,DC=arcade,DC=com")  
'Get the user profile by DN  
set objComputer= objZone.GetUserUnixProfileByDN  
("CN=yuji,CN=Users,CN=default,CN=Zones,CN=centrify,CN=program data,DC=arcade,DC=com")  
...
```



## GetUserUnixProfileByName

Returns the UNIX user profile associated with the specified user name in the zone.

### Syntax

```
IUserUnixProfile GetUserUnixProfileByName(string name)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
name	The user's UNIX login name.

### Return value

The `UserUnixProfile` object associated with the specified user name in the zone, or `null` if the `UserUnixProfile` is not found.

### Exceptions

`GetUserUnixProfileByName` may throw one of the following exceptions:

- `NotSupportedException` if the zone schema is not supported.
- `ApplicationException` if there is more than one instance of the specified user in the zone.

### Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Display the UNIX profile for the group "jae"  
wscript.Echo objZone.GetUserUnixProfileByName("jae")  
...
```

## GetUserUnixProfiles

Returns the list of UNIX user profiles for the zone.

• • • • •

## Syntax

```
IUserUnixProfiles GetUserUnixProfiles()
```

## Return value

The `UserUnixProfiles` object for the zone. This object is a collection of `UserUnixProfile` objects.

## Example

The following code sample illustrates using this method in a script to enumerate all of the user profiles in the default zone:

```
...  
'Specify the zone you want to work with  
Set objZone = cims.GetZone("acme.com/Program  
Data/Centrify/Zones/default")  
'List the UNIX login name for each profile in the zone  
Set objProfiles = objZone.GetUserUnixProfiles()  
For each profile in objProfiles  
    wscript.echo profile.Name  
next  
...
```

## GroupUnixProfileExists

Checks whether a UNIX profile exists for the specified group in the zone.

## Syntax

```
bool GroupUnixProfileExists(IGroup group)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
group	The group name for which you want to check whether a UNIX profile exists.

## Return value

Returns `true` if a UNIX profile is found in the zone for the specified group, or `false` if no UNIX profile exists for the group in the zone.



## Exceptions

`GroupUnixProfileExists` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `NotSupportedException` if the zone schema is not supported.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Check whether there's a UNIX profile for the group "legal"  
if objZone.GroupUnixProfileExists(legal) = true  
    wScript.Echo "Profile exists in this zone"  
else  
    wScript.Echo "No matching profile in this zone!"  
end if  
...
```

## LocalGroupUnixProfileExists

Checks whether a UNIX profile exists for the specified local group in the zone.

## Syntax

```
bool LocalGroupUnixProfileExists(string groupName)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>groupName</code>	The group name for which you want to check whether a UNIX profile exists.

## Return value

Returns `true` if the local UNIX group profile is found in the zone, or `false` if no UNIX profile exists for the group in the zone.

## Exceptions

`LocalGroupUnixProfileExists` may throw the following exception:



- `ArgumentNullException` if the specified parameter value is `null`.

## LocalUserUnixProfileExists

Checks whether a local UNIX profile exists for the specified user in the zone.

### Syntax

```
bool LocalUserUnixProfileExists(string userName)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>userName</code>	The local user name for which you want to check whether a UNIX profile exists.

### Return value

Returns `true` if a UNIX profile is found in the zone for the specified local user, or `false` if no UNIX profile exists for the user in the zone.

### Exceptions

`UserUnixProfileExists` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.

## PrecreateComputer

Adds a computer to a zone.

### Syntax

```
IComputer PrecreateComputer(DirectoryEntry adComputerEntry,  
string[] spn, DirectoryEntry trustee)
```

```
IComputer PrecreateCompute(DirectoryEntry containerEntry, string  
cn, string dnsName, string[] spn, DirectoryEntry trustee)
```

### Parameters

Specify the following parameters when using this method.



Parameter	Description
<code>adComputerEntry</code>	The DNS host name of the Active Directory computer object you wish to add to the zone.
<code>containerEntry</code>	The Directory container for the created computer.
<code>cn</code>	The computer name.
<code>dnsName</code>	The DNS name of the created computer.
<code>spn</code>	Service Principal Name. Specify <code>null</code> to use default.
<code>trustee</code>	The user or group to delegate <code>adjoin</code> permissions to, Specify <code>null</code> to delegate the permission for a self-service join.
<code>trusteeDn</code>	The user or group to which the computer-level overrides will be assigned, specified as a distinguished name.

## Return value

The computer object that is added to the zone.

## Discussion

Use `PrecreateComputer(DirectoryEntry, string[], DirectoryEntry)` to add an existing Active Directory computer to the zone. Use `PrecreateComputer(DirectoryEntry, string, string, string[], DirectoryEntry)` to create a new UNIX computer object and add it to the zone.

## PrecreateWindowsComputer

Adds an existing Windows computer to a zone.

## Syntax

```
IComputer PrecreateWindowsComputer(DirectoryEntry  
adComputerEntry)
```

## Parameters

Specify the following parameter when using this method.

Parameter	Description
<code>adComputerEntry</code>	The DNS host name of the Windows computer object you wish to add to the zone.

• • • • •

## Return value

The computer object that is added to the zone.

## Refresh

Reloads the zone object data from the data in Active Directory.

## Syntax

```
void Refresh()
```

## Discussion

This method refreshes the zone information in the cached object to ensure it is synchronized with the latest information in Active Directory.

## Exceptions

Refresh may throw the following exception:

- `COMException` if an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZoneByPath  
("LDAP://CN=corporate,CN=zones,CN=centrify,  
CN=program data,DC=sierra,DC=com")  
'Change the zone description  
objZone.Description = "Corporate offices, Edinburgh"  
objZone.Commit  
'Reload the zone object  
objZone.Refresh  
...
```

## UserUnixProfileExists

Checks whether a UNIX profile exists for the specified user in the zone.

• • • • •

## Syntax

```
bool UserUnixProfileExists(IUser user)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
user	The user name for which you want to check whether a UNIX profile exists.

## Return value

Returns `true` if a UNIX profile is found in the zone for the specified user, or `false` if no UNIX profile exists for the user in the zone.

## Exceptions

`UserUnixProfileExists` may throw one of the following exceptions:

- `ArgumentNullException` if the specified parameter value is `null`.
- `NotSupportedException` if the zone schema is not supported.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Check whether there's a UNIX profile for the user "garcia"  
if objZone.GroupUnixProfileExists(garcia) = true  
    wScript.Echo "Profile exists in this zone"  
else  
    wScript.Echo "No matching profile in this zone!"  
end if  
...
```

## AdsIInterface

Gets the IADs interface of the zone object from Active Directory.

## Syntax

```
IADs AdsIInterface {get;}
```

• • • • •

## Property value

The IADs interface of the zone object.

## Discussion

This property enables you to perform any operations provided by the underlying Active Directory Service Interfaces (ADSI) for a zone as a directory object. For example, you can use this property to retrieve the IADs properties and methods that enable you to access the security information for the zone object.

## Example

The following code sample illustrates using `AdsInterface` in a script:

```
...
'Specify the zone you want to work with
set zone = GetZoneByPath("LDAP://cn=test_
lab,cn=Zones,cn=UNIX,dc=ajax,dc=org")
'Get the IADs for the zone
set secdes = zone.AdsInterface.Get("ntSecurityDescriptor")
'Display security information for the zone
wScript.Echo secdes.Owner
...
```

## ADsPath

Gets the LDAP path to the zone object.

## Syntax

```
string ADsPath {get;}
```

## Property value

The full LDAP path to the zone object.

## Example

The following code sample illustrates using `ADsPath` in a script:

```
...
'Specify the zone you want to work with
set zone = GetZone("fireball.net/Field/Zones/macs")
'Display the LDAP path for the zone
wScript.Echo zone.ADsPath
...
```



## AgentlessAttribute

Gets or sets the Active Directory attribute used for storing the user's password hash if a zone supports agentless NIS client requests.

### Syntax

```
string AgentlessAttribute {get; set;}
```

### Property value

The Active Directory attribute used for storing the user's password hash.

### Discussion

If you have any computers configured to respond to NIS client requests using information stored in Active Directory, this property must be set to enable password synchronization for the zone. Only the following values are valid:

- `altSecurityIdentities`
- `mSsfu30Password`
- `unixUserPassword`

Setting this property to an invalid value disables password synchronization.

### Exceptions

`AgentlessAttribute` throws an `ApplicationException` if the selected attribute cannot store a password hash.

### Example

The following code sample illustrates setting this property in a script:

```
...  
'Specify the zone you want to work with  
set zone = cims.GetZone("zap.org/Program  
Data/Centrify/Zones/default")  
'Change the attribute used for the password hash  
zone.AgentlessAttribute = "unixUserPassword"  
zone.Commit()  
...
```



## AvailableShells

Gets or sets the list of available shells for this zone.

### Syntax

```
string[ ] AvailableShells {get; set;}
```

### Property value

The list of available shells for the zone.

### Discussion

The values you define for this property are used as the values in the drop-down list of available shells when defining the UNIX profile for a new user in the Access Manager console.

This property requires a strongly-typed array. Because strongly-typed arrays are not supported in VBScript, you cannot use this property in scripts written with VBScript. To use this property, you must use a programming language that allows strongly-typed arrays.

### Example

The following code sample illustrates setting this property in a Visual Studio (C#) script:

```
...
// Set the Active Directory container object.
DirectoryEntry objContainer = new DirectoryEntry
("LDAP://cn=Zones,cn=UNIX,dc=ajax,dc=org");
IZone objZone = cims.CreateZone(objContainer, "QA Zone");

// set the starting UID and GID for the zone
objZone.NextAvailableUID = 10000;
objZone.NextAvailableGID = 10000;

// set the list of available shells and default shell for the
zone
objZone.AvailableShells = new string[] {"/bin/bash",
"/bin/shell"};
objZone.DefaultShell = "/bin/bash";

// set the default home directory for the zone
objZone.DefaultHomeDirectory = "/home/${user}";
objZone.Commit();
Console.WriteLine("Zone created successfully.");
...
```



## Cims

Gets the Cims object managing the zone.

### Syntax

```
Cims Cims {get;}
```

### Property value

The Cims object managing this zone.

### Discussion

This property serves as a shortcut for retrieving data.

## DefaultGroup

Gets or sets the default group profile to use as the primary group for new users in the zone.

### Syntax

```
IGroupUnixProfile DefaultGroup{get; set;}
```

### Property value

The default group property for the zone.

### Discussion

The default group profile for a zone is always associated with an existing Active Directory group. You can, however, define a primary group that is not associated with any Active Directory group.

For more information about defining primary groups for UNIX users, see the *Planning and Deployment Guide* and the *Administrator's Guide for Linux and UNIX*.

### Example

The following code sample illustrates using `DefaultGroup` in a script:

• • • • •

```
...
'Specify the zone you want to work with
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")
'Check the default group for the zone
If objZone.DefaultGroup is nothing
  'Identify the Active Directory group object
  set objGrp = cims.GetGroupByPath("LDAP://CN=IT
Interns,CN=Users,DC=ajax,DC=org")
  'Get the UNIX profile for the Active Directory group
  set objGrpProfile = objZone.GetGroupUnixProfile(objGrp)
  'Make this profile the default group
  set objZone.DefaultGroup = objGrpProfile
end if
objZone.Commit
...
```

## DefaultHomeDirectory

Gets or sets the local file system path to the user's default home directory.

### Syntax

```
string DefaultHomeDirectory {get; set;}
```

### Property value

The text string that defines the default path to the user's home directory.

### Discussion

The only variable permitted is `${user}`. The Access Manager console replaces this variable with the user's UNIX login name when you add a UNIX profile for the user to the zone.

### Example

The following code sample illustrates using `DefaultHomeDirectory` in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZone("zap.org/Program
Data/Centrify/Zones/default")
'Set zone properties
objZone.DefaultHomeDirectory = "/home/${user}"
objZone.DefaultShell = "/bin/bash"
objZone.NextAvailableUID = zone.NextAvailableUID + 1
objZone.NextAvailableGID = zone.NextAvailableGID + 1
objZone.DefaultHomeDirectory = "/home/${user}"
...
```

• • • • •

## DefaultShell

Gets or sets the default shell assigned to new users in the zone.

### Syntax

```
string DefaultShell {get; set;}
```

### Property value

The default shell property for the zone.

### Discussion

The value you define for this property is automatically populated as the default shell when defining the UNIX profile for a new user in Access Manager. The value can be overridden by the administrator defining the user's profile.

### Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("zap.org/Program  
Data/Centrify/Zones/default")  
'Specify zone properties  
objZone.DefaultHomeDirectory = "/home/admin"  
objZone.DefaultShell = "/bin/bash"  
objZone.NextAvailableUID = zone.NextAvailableUID + 1  
objZone.NextAvailableGID = zone.NextAvailableGID + 1  
...
```

## DefaultValueZone

Gets or sets the zone to use as the "master" zone for setting default zone property values.

### Syntax

```
IZone DefaultValueZone {get; set;}
```

### Property value

The zone object for the zone used to define default values.



## Discussion

If this property is set, the profile information and zone properties in the specified zone are used as the default values for the current zone. For example, if you add users or groups that have profiles in the specified zone to the zone context in which you are currently working, their UNIX profiles have the same UIDs and GIDs in both zones by default.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Get the master default values zone for this zone  
set objMaster = objZone.DefaultValueZone  
wScript.Echo "The master zone is " & objMaster.Name  
...
```

## Description

Gets or sets the text string used for the description property of the zone.

## Syntax

```
string Description {get; set;}
```

## Property value

The description property for the zone.

## Discussion

The zone description can consist of any text string up to the number maximum of characters available in the data attribute where zone properties are stored. The maximum number of characters available for the attribute is approximately 850, but the maximum available for the description depends on the other data being stored. For example, the more available shells you have defined for a zone, the shorter the zone description must be.

## Example

The following code sample illustrates setting this property in a script:

• • • • •

```
...  
'Specify the zone you want to work with  
set zone = cims.GetZone("fireball.net/Field/Zones/macs")  
'Set the long description for the zone  
zone.Description = "Mac OS X computers and users in Fireball  
field offices"  
zone.Commit()  
...
```

## FullName

Gets the full name of the zone.

### Syntax

```
string FullName {get;}
```

### Property value

The full, canonical name for the zone.

### Discussion

The full name is the canonical name of the zone. The full name is updated when the directory object is updated. Therefore, changes to the **Name** property are not reflected in the value retrieved by the **FullName** property until the changes to the **Name** property are committed to Active Directory.

### Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Display the full name zone  
If objZone.IsReadable = true  
    wScript.Echo "Zone name: " & objZone.FullName  
end if  
...
```

## GroupAutoProvisioningEnabled

Indicates whether auto-provisioning of group profiles is enabled for the zone.

• • • • •

## Syntax

```
bool GroupAutoProvisioningEnabled {get;}
```

## Property value

Returns true if the zone has auto-provisioning enabled.

## Discussion

When automatic provisioning is enabled for groups, the Zone Provisioning Agent can automatically provision new UNIX profiles for groups added to the zone. In addition to enabling provisioning, you must specify a provisioning group to use as the source for provisioning data. For more information about automatic provisioning of users and groups, see the *Planning and Deployment Guide*.

## ID

Gets the unique identifier for the zone.

## Syntax

```
string ID {get;}
```

## Property value

The unique identifier for the zone.

## Discussion

This property is used internally to prevent a zone from being listed more than once.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set zone = GetZoneByPath("LDAP://cn=test_  
lab,cn=Zones,cn=UNIX,dc=ajax,dc=org")  
'Display the unique identifier for the zone  
wScript.Echo zone.ID  
...
```



## IsHierarchical

Indicates whether the zone supports hierarchical zone features.

### Syntax

```
bool IsHierarchical {get;}
```

### Property value

Returns true if the zone is hierarchical.

## IsReadable

Determines whether this zone object's properties are readable for the user whose credentials are presented.

### Syntax

```
bool IsReadable {get;}
```

### Property value

Returns true if the zone object is readable by the user, or false if the zone object is not readable.

## Discussion

This property returns a value of true if the user accessing the zone object in Active Directory has sufficient permissions to read zone properties.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Check whether the zone is readable  
If not objZone.IsReadable then  
    wScript.Echo "You do not have read permission for this zone"  
end if  
...
```



## IsSFU

Determines whether the zone uses the Microsoft Services for UNIX (SFU) schema extension.

### Syntax

```
bool IsSFU {get;}
```

### Property value

Returns `true` if the zone uses a Services for UNIX (SFU) schema, or `false` if the zone does not use the SFU schema.

### Discussion

If the Microsoft Services for UNIX (SFU) schema extension is installed and being used to store UNIX attributes for the zone, this property returns a value of `true`.

### Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set zone = GetZoneByPath("LDAP://cn=test_  
lab,cn=Zones,cn=UNIX,dc=ajax,dc=org")  
'Check whether the zone uses the SFU schema  
If zone.IsSFU then  
    wScript.Echo "Zone uses the SFU schema for UNIX attributes"  
end if  
...
```

## IsTruncateName

Determines whether the zone is a TruncateName zone.

### Syntax

```
bool IsTruncateName {get; set;}
```

### Property value

If this property is `true`, the zone is a TruncateName zone.

• • • • •

## Discussion

If this property is `true`, the default pre-Windows 2000 name for new users is the computer account name truncated at 15 characters.

## IsWritable

Determines whether the zone object's properties are writable properties for the user whose credentials are presented.

## Syntax

```
bool iswritable {get;}
```

## Property value

Returns `true` if the zone object is writable by the user, or `false` if the zone object is not writable.

## Discussion

This property returns a value of `true` if the user accessing the zone object in Active Directory has sufficient permissions to change zone properties.

## Example

The following code sample illustrates using this property in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZone("ajax.org/UNIX/Zones/test_lab")  
'Check whether the zone is writable  
If not objZone.IsWritable then  
    wscript.Echo "You do not have permission to modify the zone"  
end if  
...
```

## Licenses

Gets or sets the license container associated with this zone.

## Syntax

```
ILicenses Licenses {get; set;}
```

• • • • •

## Property value

The license container for this zone.

## MasterDomainController

Gets or sets the name of the domain controller to use as the primary domain controller for the zone.

## Syntax

```
string MasterDomainController {get; set;}
```

## Property value

The name of the primary domain controller for the zone.

## Example

The following code sample illustrates using `MasterDomainController` in a script:

```
...  
'Specify the zone you want to work with  
set zone = GetZone("fireball.net/Field/Zones/macs")  
'Display the domain controller for this zone  
wScript.Echo "Main Domain Controller: " &  
zone.MasterDomainController  
...
```

## MustMaintainADGroupMembership

Gets or sets the flag that indicates whether Active Directory group membership must be maintained.

## Syntax

```
bool MustMaintainADGroupMembership {get; set;}
```

## Property value

Returns `true` if the Active Directory group membership must be maintained; otherwise, it returns `false`.



## Discussion

By default, setting the primary Active Directory group in a user's UNIX profile does not affect the user's actual Active Directory group membership.

If you want to enforce Active Directory group membership for new users when you add them to the zone, set this property to `true`. Setting this property to `true` displays the **Associate Active Directory group membership** option in the Zone Properties dialog box.

## Example

The following code sample illustrates using `MustMaintainADGroupMembership` in a script:

```
...
'Specify the zone you want to work with
set zone = GetZone("fireball.net/Field/Zones/macs")
'Check whether Active Directory group membership is enforced
if zone.MustMaintainADGroupMembership then
    wScript.Echo "Active Directory group membership maintained"
else
    wScript.Echo "No Active Directory group membership needed"
end if
...
```

## Name

Gets or sets the name of the zone.

## Syntax

```
string name {get; set;}
```

## Property value

The short name of the zone. The name must start with an alphanumeric character or an underscore (`_`) character and can contain any combination of letters (upper- or lowercase), numerals 0 through 9, and the period (`.`), hyphen (`-`) and underscore (`_`) characters up to a maximum length of 64 characters.

## Exceptions

`Name` throws an `ArgumentException` if you try to set an invalid name for the zone.



## Example

The following code sample illustrates setting this property in a script:

```
...  
'Specify the zone you want to work with  
set zone = cims.GetZone("zap.org/Program  
Data/Centrify/Zones/default")  
'Change the name of the "default" zone to "Pilot deployment"  
zone.Name = "Pilot deployment"  
zone.Commit()  
...
```

## NextAvailableGID

Returns or sets the next available value for the group identifier (GID) in the zone.

### Syntax

```
int NextAvailableGID {get; set;}
```

### Property value

The numeric value of the next available GID for the zone.

### Discussion

This method returns or sets the next available GID to be used as the default GID assignment for the next group given access to the zone. If you are setting this property as part of creating a new zone, use this value to define the starting GID value for all groups in the zone. In most cases, this value is incremented automatically each time a new group profile is created for the zone. If you are creating new groups programmatically, use this property to read the current value.

There are two versions of this property: one designed for COM-based programs (`NextAvailableGID`) that supports a 32-bit signed number for the GID and one designed for .NET-based programs (`NextGID`) that allows a 64-bit signed number for the GID. You can use either property.

## Example

The following code sample illustrates setting this property in a script:

• • • • •

```
...  
'Set the container object for the zone  
set objContainer = GetObject("LDAP://cn=Zones,cn=UNIX,  
dc=ajax,dc=org")  
'Create a new zone named "Sample_Zone"  
set objZone = cims.CreateZone(objContainer, "Sample_Zone")  
'Set the starting UID and GID for the new zone  
objZone.nextAvailableUID = 10000  
objZone.nextAvailableGID = 10000  
...
```

## NextAvailableUID

Returns or sets the next available value for the user identifier (UID) in the zone.

### Syntax

```
int nextAvailableUID {get; set;}
```

### Property value

The numeric value of the next available UID for the zone.

### Discussion

This method returns or sets the next available UID to be used as the default UID assignment for the next user given access the zone. If you are setting this property as part of creating a new zone, use this value to define the starting UID for all users in the zone. In most cases, this value is incremented automatically each time a new user is enabled for the zone. If you are creating new users programmatically, you can use this property to read the current value.

There are two versions of this property: one designed for COM-based programs (`nextAvailableUID`) that supports a 32-bit signed number for the UID and one designed for .NET-based programs (`NextUID`) that allows a 64-bit signed number for the UID. You can use either method.

### Example

The following code sample illustrates setting this property in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cdc.GetZone("ajax.org/UNIX/Zones/ea_central")  
'Reset the next available UID for the zone  
objZone.nextAvailableUID = 5000
```



```
zone.Commit()  
...
```

## NextGID

Gets or sets the next GID to be used when adding groups (64-bit for use with .NET).

### Syntax

```
long NextGID {get; set;}
```

### Property value

The GID for new groups.

### Discussion

This method returns or sets the next available GID to be used as the default GID assignment for the next group given access to the zone. If you are setting this property as part of creating a new zone, use this value to define the starting GID value for all groups in the zone. In most cases, this value is incremented automatically each time a new group profile is created for the zone. If you are creating new groups programmatically, use this property to read the current value.

There are two versions of this property: one designed for COM-based programs ([NextAvailableGID](#)) that supports a 32-bit signed number for the GID and one designed for .NET-based programs ([NextGID](#)) that allows a 64-bit signed number for the GID. You can use either method.

## NextUID

Gets or sets the next UID to be used when adding users (64-bit for use with .NET).

### Syntax

```
long NextUID {get; set;}
```



## Property value

The UID for new users.

## Discussion

This method returns or sets the next available UID to be used as the default UID assignment for the next user given access to the zone. If you are setting this property as part of creating a new zone, use this value to define the starting UID for all users in the zone. In most cases, this value is incremented automatically each time a new user is enabled for the zone. If you are creating new users programmatically, you can use this property to read the current value.

There are two versions of this property: one designed for COM-based programs (**NextAvailableUID**) that supports a 32-bit signed number for the UID and one designed for .NET-based programs (**NextUID**) that allows a 64-bit signed number for the UID. You can use either method.

## NISDomain

Gets or sets the NIS domain associated with the zone when the zone is determined to be a zone that uses the Microsoft Services for UNIX (SFU) schema extension or is configured to support agentless NIS client requests.

## Syntax

```
string NISDomain {get; set;}
```

## Property value

The Network Information Service (NIS) distinguished name for the zone.

## Discussion

If the zone is a Services for UNIX (SFU) zone, this property should be the NIS domain defined in users' UNIX attributes. For agentless client requests, the zone associated with the computer acting as the NIS server is the NIS domain.

## Exceptions

**NISDomain** throws an **ApplicationException** if no value is specified when setting this property. You must specify a value when using this property to set

• • • • •

the NIS domain.

## Example

The following code sample illustrates using this method in a script:

```
...
'Specify the zone you want to work with
set zone = GetZoneByPath("LDAP://cn=test_
lab,cn=Zones,cn=UNIX,dc=ajax,dc=org")
'If the zone uses the SFU schema, display its NIS domain
If zone.IsSFU then
    wScript.Echo "NIS Domain: " & zone.NISDomain
end if
...
```

## ReservedGID

Gets or sets the list of reserved group identifiers (GIDs) in the zone.

### Syntax

```
string[ ] ReservedGID {get; set;}
```

### Discussion

Reserved GIDs cannot be assigned when creating new groups. The `get` argument returns a string containing the range of GIDs not available. The `set` argument specifies a number range to be reserved.

This property requires a strongly-typed array. Because strongly-typed arrays are not supported in VBScript, you cannot use this property in scripts written with VBScript. To use this property, you must use a programming language that allows strongly-typed arrays.

## Example

The following code sample illustrates using `ReservedUID` in a Visual Studio (C#) script:

```
...
IZone objZone = cims.CreateZone(objContainer, strZone);
// Set the starting UID and GID for the zone
objZone.NextAvailableUID = 10000;
objZone.NextAvailableGID = 10000;

// Set the reserved UIDs and GIDs for the zone
objZone.ReservedUID = new string[] {"0-300", "999"};
```

• • • • •

```
objZone.ReservedGID = new string[] {"0-300", "999"};  
objZone.Commit();  
...
```

## ReservedUID

Gets or sets the list of reserved user identifiers (UIDs).

### Syntax

```
string[ ] ReservedUID {get; set;}
```

### Discussion

Reserved UIDs cannot be assigned when creating new users. The `get` argument returns a string containing the range of UIDs not available. The `set` argument specifies a number range to be reserved.

This property requires a strongly-typed array. Because strongly-typed arrays are not supported in VBScript, you cannot use this property in scripts written with VBScript. To use this property, you must use a programming language that allows strongly-typed arrays.

### Example

The following code sample illustrates using `ReservedUID` in a Visual Studio (C#) script:

```
...  
IZone objZone = cims.CreateZone(objContainer, strZone);  
// Set the starting UID and GID for the zone  
objZone.NextAvailableUID = 10000;  
objZone.NextAvailableGID = 10000;  
  
// Set the reserved UIDs and GIDs for the zone  
objZone.ReservedUID = new string[] {"0-300", "999"};  
objZone.ReservedGID = new string[] {"0-300", "999"};  
objZone.Commit();  
...
```

### Schema

Gets the schema type of the zone object.



## Syntax

ZoneSchema Schema {get;}

## Property value

The schema type for the zone.

## Discussion

The schema type defines how data for the zone should be stored in Active Directory and is based on the specific Active Directory schema you are using. Zones can be defined as:

- Standard Centrify zones
- Standard Centrify RFC 2307-compliant zones
- Centrify Services for UNIX (SFU) zones

The schema type provides an additional level of granularity corresponding the specific version of the Active Directory schema you are using and where specific zone properties and UNIX attributes are stored. The schema types currently defined for Centrify zones are:

Schema name	Value	Description
Unknown	-1	Schema unknown
Dynamic_Schema_1_0	0	Standard Centrify zone, version 1.x Uses the Centrify version 1.x and standard Active Directory schema data storage model. This zone type is for backward compatibility and otherwise no longer in use.
Dynamic_Schema_2_0	1	Standard Centrify zone, version 2.x and 3.x Uses the Centrify version 2.x and standard Active Directory schema data storage model. This zone type is for backward compatibility and otherwise no longer in use.
SFU_3_0	2	SFU zone, version 2.x and 3.x Uses a combination of the Centrify version 3.x and Microsoft Services for UNIX (SFU) 3.0 data storage model. This zone type can be used when Active Directory has the Microsoft Services for UNIX (SFU), version 3.x, schema extension installed. The standard UNIX properties are stored as defined by the Microsoft SFU 3.x schema, but associated with zones. This zone type is for backward compatibility if you have the Microsoft Services for UNIX (SFU) schema extension installed, and otherwise no longer in use.



Schema name	Value	Description
		SFU zone, version 4.x
SFU_4_0	3	Uses a combination of the Centrify version 3.x and Microsoft Services for UNIX (SFU) 4.0 data storage model. This zone type can be used when Active Directory has the Microsoft Services for UNIX (SFU), version 4.0, schema extension installed. The standard UNIX properties are stored as defined by the Microsoft SFU 4.0 schema, but associated with zones.  This zone type is for backward compatibility if you have the Microsoft Services for UNIX (SFU) schema extension installed, and otherwise no longer in use.
CDC_RFC_2307	5	Standard RFC 2307-compatible zone, version 3.x  Uses the Active Directory RFC 2307-compliant schema data storage model.
Dynamic_Schema_3_0	6	Standard Centrify zone, version 3.x and 4.x  Uses the Centrify version 4.x and Active Directory schema data storage model.  <b>Note</b> The only difference between the <code>Dynamic_Schema_2_0</code> data storage model and the <code>Dynamic_Schema_3_0</code> data storage model is the use of the <code>managedBy</code> attribute. This attribute is set in zones that use the <code>Dynamic_Schema_2_0</code> schema. The <code>managedBy</code> attribute is not used in zones that use in the <code>Dynamic_Schema_3_0</code> schema.
CDC_RFC_2307_2	7	Classic RFC 2307-compatible zone, version 4.x  Uses the Active Directory RFC 2307-compliant schema data storage model.  <b>Note</b> The only difference between the <code>CDC_RFC_2307</code> data storage model and the <code>CDC_RFC_2307_2</code> data storage model is the use of the <code>managedBy</code> attribute. This attribute is set in zones that use the <code>CDC_RFC_2307</code> schema. The <code>managedBy</code> attribute is not used in zones that use in the <code>CDC_RFC_2307_2</code> schema.
Dynamic_Schema_5_0	8	Hierarchical zone, version 5.x  Uses the Centrify version 5.x and standard Active Directory schema data storage model.  <b>Note</b> The difference between the <code>Dynamic_Schema_5_0</code> data storage model and the <code>CDC_RFC_2307_3</code> data storage model is that in the <code>Dynamic_Schema_5_0</code> storage model, all Centrify data is stored as part of the zone. In the <code>CDC_RFC_2307_3</code> storage model, user and group attributes are stored as part of the <code>User</code> and <code>Group</code> objects.
CDC_RFC_2307_3	9	Hierarchical RFC 2307-compatible zone, version 5.x
SFU_3_0_v5	10	Hierarchical SFU zone, version 5.x



If the zone is not in one of these formats, an exception is thrown. For more information about the difference between these different schema types and the corresponding zone types, see “Planning for data storage in Active Directory” in the *Planning and Deployment Guide*.

## Exceptions

Schema throws an `ApplicationException` if the zone schema is not recognized.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set zone = GetZone("ajax.org/UNIX/Zones/test_lab")  
'If the zone uses the SFU schema, display its domain  
If zone.IsSFU = true  
    wScript.Echo zone.SFUDomain  
end if  
...
```

## SFUDomain

Gets or sets the Active Directory domain associated with this zone when the zone is determined to be a zone that uses the Microsoft Services for UNIX (SFU) schema extension.

## Syntax

```
string SFUDomain {get; set;}
```

## Property value

The Active Directory domain for the zone.

## Example

The following code sample illustrates using this method in a script:

```
...  
'Specify the zone you want to work with  
set zone = GetZoneByPath("LDAP://cn=test_  
lab,cn=Zones,cn=UNIX,dc=ajax,dc=org")  
'If the zone uses the SFU schema, display its domain  
If zone.IsSFU = true
```

• • • • •

```
wScript.Echo zone.SFUDomain  
end if  
...
```

## UserAutoProvisioningEnabled

Indicates whether the zone has auto-provisioning of user profiles enabled.

### Syntax

```
bool UserAutoProvisioningEnabled {get;}
```

### Property value

Returns true if the zone has auto-provisioning enabled for user profiles.

### Discussion

When automatic provisioning is enabled for users, the Zone Provisioning Agent can automatically provision new UNIX profiles for new Active Directory users. In addition to enabling auto-provisioning, you must specify a provisioning group to use as the source for provisioning data. For details about automatic provisioning of users and groups, see the *Planning and Deployment Guide*.

## Version

Gets the version number of the data schema.

### Syntax

```
int version {get;}
```

### Property value

The version number associated with the schema found.

### Example

The following code sample illustrates using `version` in a script:

```
...  
'Specify the zone you want to work with  
set zone = GetZoneByPath("LDAP://cn=test_lab/cn=Zones,  
cn=UNIX,dc=ajax,dc=org")
```



```
'Display the schema version number for the zone  
wScript.Echo zone.Version  
...
```

## Entry

The Entry class contains methods and properties used to manage individual NIS map entries stored in Active Directory. This class is defined in the `Centrify.DirectControl.NISMap.API` namespace rather than the `Centrify.DirectControl.API` namespace.

## Syntax

```
public class IEntry : ICloneable, IDisposable
```

## Discussion

Each map entry consists of three primary fields: a key field, a value field, and an optional comment field.

The Entry class supports the methods and properties that apply to all .NET objects. In addition to those methods and properties, the Entry class provides some Centrify-specific methods and properties for managing the fields in NIS map records. Only the Centrify-specific methods and properties are described in this reference.

## Methods

The Entry class provides the following Centrify-specific methods:

This method	Does this
<code>Clone</code>	Makes a clone of the NIS map entry. Inherited from <code>ICloneable</code> .
<code>Commit</code>	Commits changes to the NIS map entry object and saves them in Active Directory.
<code>Dispose</code>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.



This method	Does this
	Inherited from <code>IDisposable</code> .
<code>GetDirectoryEntry</code>	Returns the <code>DirectoryEntry</code> attribute for the NIS map entry object.

## Properties

The `Entry` class provides the following Centrify-specific properties:

This property	Does this
<code>Comment</code>	Gets or sets the comment field associated with a specific key in a map entry.
<code>IsReadable</code>	Indicates whether the map entry is readable.
<code>IsWritable</code>	Indicates whether the map entry is writable.
<code>Key</code>	Gets or sets the key field in a map entry.
<code>Map</code>	Gets the NIS map associated with the map entry.
<code>Value</code>	Gets or sets the value field associated with a specific key in a map entry.

## Commit

Commits the settings or changes for the map entry object to Active Directory.

## Syntax

```
void Commit();
```

## Exceptions

`Commit` throws an `ApplicationException` if it can't find the `DirectoryEntry` value or if the key or value is invalid.

## Example

The following code sample illustrates using `Commit` to make changes to an existing NIS map entry to Active Directory:

```
...
'Identify the zone you want to work with
set zone = cims.GetZone("sample.com/centrify/zones/default")
'Create the Store object
set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and username and password.
```



```
store.Attach zone.ADSPATH, "jae.smith", "pas$w0rd"  
'Open the generic map type named "Workstations IDs"  
Set map = store.open("workstations IDs")  
'Modify the value field for the "workstation" map entry:  
set entry = map.get("128.10.12.1")  
entry.Value = "satellite1"  
'Commit the changes to Active Directory  
entry.Commit  
wScript.Echo "NIS map entry " & entry.Key & ": " & entry.Value  
...
```

## GetDirectoryEntry

Returns the DirectoryEntry object for the map entry object.

### Syntax

```
DirectoryEntry GetDirectoryEntry ()
```

### Return value

The directory entry for the map entry object.

### Discussion

This method can only be used in .NET programs because DirectoryEntry is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

## Comment

Gets or sets the comment field for a specific NIS map entry.

### Syntax

```
string Comment {get; set;}
```

### Property value

The contents of the comment field for a specific NIS map entry.



## Discussion

Each map entry consists of three primary fields: a key field, a value field, and an optional comment field. To use this property, you must be able to identify the map and the entry—the specific record in the map—for which you are setting or retrieving the comment.

## Exceptions

`Comment` throws an `ArgumentException` if you try to set a value greater than 2048 characters.

## Example

The following code sample illustrates using `Comment` to change the comment field in an existing NIS map entry:

```
...
'Identify the zone you want to work with
set zone = cims.GetZone("sample.com/centrify/zones/default")
'Create the store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and username and password.
store.Attach zone.ADsPath, "jae.smith", "pas$w0rd"
'Open the generic map type named "Workstations IDs"
Set map = store.open("workstations IDs")
'Modify the Comment field for workstation "128.10.12.1" map
entry:
set entry = map.get("128.10.12.1")
entry.Comment = "San Francisco, 5th floor, Accounting Dept."
'Commit the changes to Active Directory
entry.Commit
...
```

## IsReadable

Indicates whether the map entry is readable for the user credentials presented to connect to Active Directory.

## Syntax

```
bool IsReadable {get;}
```

• • • • •

## Property value

Returns `true` if the map entry object is readable by the user, or `false` if the map entry object is not readable.

## Discussion

This property returns a value of `true` if the user accessing the map entry object in Active Directory has sufficient permissions to read the entry properties.

## Example

The following code sample illustrates using this property in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath("LDAP://CN=qa-
slovenia,CN=unix,DC=quantum,DC=net")
'Create the store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and username and password.
store.Attach objZone.ADSPATH, "jae.smith", "pas$w0rd"
'Open the generic map type named "Workstations IDs"
Set map = store.open("workstations IDs")
'Get the map entry specified
Set entry = map.get("128.10.12.1")
'Check whether the record is readable
If not entry.IsReadable then
    wscript.Echo "No read permission for this record"
end if
...
```

## IsWritable

Indicates whether the map entry is writable for the user credentials presented to connect to Active Directory.

## Syntax

```
bool Iswritable {get;}
```

## Property value

Returns `true` if the map entry object is writable by the user, or `false` if the map entry object is not writable.



## Discussion

This property returns a value of `true` if the user accessing the map entry object in Active Directory has sufficient permissions to change the entry object's properties.

## Example

The following code sample illustrates using this property in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath("LDAP://CN=qa-
slovenia,CN=unix,DC=quantum,DC=net")
'Create the Store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and username and password.
store.Attach objZone.ADsPath, "jae.smith", "pas$w0rd"
'Open the generic map type named "Workstations IDs"
Set map = store.open("workstations IDs")
'Get the map entry specified
Set entry = map.get("128.10.12.1")
'Check whether the record is writable
If not entry.IsWritable then
    wScript.Echo "No write permission for this record"
end if
...
```

## Key

Gets or sets the key field for a NIS map entry.

## Syntax

```
string Key {get; set;}
```

## Property value

The contents of the key field for a NIS map entry.

## Discussion

Each map entry consists of three primary fields: a key field, a value field, and an optional comment field.



## Exceptions

Key throws an `ArgumentException` if you try to set a value that is null, empty, or greater than 1024 characters.

## Example

The following code sample illustrates using `Key` to make changes to an existing NIS map entry and commit the changes to Active Directory.

```
...
'Identify the zone you want to work with
set zone = cims.GetZone("sample.com/centrify/zones/default")
'Create the Store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and user credentials (username and
'password).
store.Attach zone.ADsPath, "jae.smith", "pas$w0rd"
'Open the NIS map named "generic map"
Set map = store.open("generic map")
'Modify the map entry fields for the "Key_Name" map record:
'entry.key = Key_Name
'entry.Value = Key_Value
'entry.comment = This is a sample generic map entry"
set entry = map.get("Key_Name")
entry.Key      = "Modified_Key"
entry.Value    = "Modified_Value"
entry.Comment  = "Modified comment for the sample map entry"
'Commit the changes to Active Directory
entry.Commit
wScript.Echo "NIS map entry has been modified."
...
```

## Map

### Syntax

```
Map Map {get;}
```

### Property value

The map containing this entry.

## Value

Gets or sets the value field associated with a specific NIS map entry key.



## Syntax

```
string value {get; set;}
```

## Property value

The value field associated with a specific NIS map entry key.

## Discussion

Each map entry consists of three primary fields: a key field, a value field, and an optional comment field. The content and format of the value field depends on the type of NIS map you are working with. For example, if you are setting the value field in a generic map, the field can contain virtually any string that you want served for a corresponding key name.

If you are setting the value field in a `netgroup` map, the field lists the members of the group, separated by a blank space. Each member can be either a group name or a triple of the form (hostname, username, domainname). For example:

```
set map = store.open("netgroup")
set entry = map.get("db_users")
entry.Value = "fin hr (,dean,ajax.org) (clone*,,)"
```

If you are defining the value field in an `auto.mount` map entry, the field consists of the mount options, a tab character, and the network path to the file to consult for the mount point being defined. For example:

```
set map = store.open("auto.mount")
'Modify the value field of the "cdrom" mount point entry
set entry = map.get("cdrom")
entry.Value = "-fstype=nsfs,ro" & Chr(11) & ":/dev/sr0"
```

If you are defining the value field in an `auto.master` map entry, the field consists of the map file to consult, a tab character, and the mount options for the mount point being defined. For example:

```
set map = store.open("auto.mount")
'Modify the value field of the "/net" mount point entry
set entry = map.get("/net")
entry.Value = "-hosts" & Chr(11) & "-nosuid,nobrowse"
```

## Exceptions

`Value` throws an `ArgumentException` if you try to set a value that is null, empty, or greater than 1024 characters.



## Example

The following code sample illustrates using `Value` to set the value associated with a specified NIS map entry in a `netgroup` map:

```
...
'Identify the zone you want to work with
set zone = cims.GetZone("sample.com/centrify/zones/default")
'Create the Store object
set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone.
'Provide the path to the zone and user credentials
store.Attach zone.ADSPATH, "jae.smith", "pas$w0rd"
'Open the NIS map named "netgroup"
set map = store.open("netgroup")
'Modify the value field of the "db_admins" entry
set entry = map.Get("db_admins")
entry.Value = "dbas dbowners (,dean,) (firebird,jon,)"
...
```

## Map

The `Map` class contains methods and properties used to manage individual NIS map entries stored in Active Directory. This class is defined in the `Centrify.DirectControl.NISMap.API` namespace rather than the `Centrify.DirectControl.API` namespace.

## Syntax

```
public class IMap : ICloneable, IDisposable
```

## Discussion

The `Map` class supports the methods and properties that apply to all .NET objects. In addition to those methods and properties, the `Map` class provides some Centrify-specific methods and properties for managing individual NIS map records. Only the Centrify-specific methods and properties are described in this reference.

## Methods

The `Map` class provides the following Centrify-specific methods:



This method	Does this
<b>Add</b>	Adds a new map entry with the specified key to the NIS map.
<b>Clone</b>	Makes a clone of the NIS map entry. Inherited from <b>ICloneable</b> .
<b>Commit</b>	Commits changes to the NIS map and saves them in Active Directory.
<b>Dispose</b>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from <b>IDisposable</b> .
<b>Exists</b>	Checks whether there is an entry with a specific key-value pair.
<b>Get</b>	Returns the NIS map entry with the specified key.
<b>GetByID</b>	Returns the NIS map entry with the specified ID.
<b>GetDirectoryEntry</b>	Returns the <b>DirectoryEntry</b> for the NIS map object.
<b>GetEnumerator</b>	Returns an enumeration of <b>IMap</b> objects.
<b>GetRedirectMap</b>	Returns the redirect target NIS map.
<b>Import</b>	Imports a new map entry with the specified key.
<b>Remove</b>	Removes the map entry with the specified key.
<b>RemoveByID</b>	Removes the map entry with the specified ID.

## Properties

The Map class provides the following Centrify-specific properties:

This property	Does this
<b>IsReadable</b>	Indicates whether the map object is readable.
<b>IsWritable</b>	Indicates whether the map object is writable.
<b>Name</b>	Gets or sets the map name of the NIS map.
<b>Store</b>	Gets the Store instance associated with the map object.
<b>Type</b>	Gets or sets the NIS map type.

## Add

Adds a new map entry, or individual map record, with the specified key.

## Syntax

```
Entry Add(string key, string value, string comment);  
Entry Add(string key);
```



## Parameters

Specify the following parameters when using this method.

Parameter	Data type	Description
key	String	The key field that uniquely defines this entry in the NIS map.
value	String	The value associated with the key field for this entry in the NIS map.
comment	String	Any optional text string to store in the comment field for this entry in the NIS map.

## Return value

An entry object instance for the map entry added.

## Discussion

Each map entry consists of three primary fields: a key field, a value field, and an optional comment field. The content and format of the value field depends on the type of NIS map you are working with. For example, if you are setting the value field in a generic map, the field can contain virtually any string that you want served for a corresponding key name. If you are adding a map entry to a `netgroup`, `auto.mount`, or `auto.master` map, the single value field defined in Active Directory may consist of multiple, concatenated fields appropriate for the map type.

Because of potential API name conflict, the `Add(string key)` method can be used with .NET programs only.

## Example

The following code sample illustrates using `Add` to add new NIS entries to different types of NIS maps:

```
'''
'Identify the zone you want to work with
set zone = cims.GetZone("sample.com/centrify/zones/default")
'Create the Store object
set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone.
'Provide the path to the zone and user credentials (username and
'password).
store.Attach zone.ADsPath, "jae.smith", "pas$w0rd"
'Open the generic NIS map named "Remote servers"
set map = store.open("Remote servers")
'Add an entry to the "Remote servers" NIS map. The input format
is:
'map.add "<key>", "<value>", "<comment>"
```



```
map.add "mirage", "127.67.10.1", "Server located in Toledo
office"
'Open the NIS map named "auto.master"
Set map = store.open("auto.master")
'Add an entry to the "auto.master" NIS map. The input format is:
'map.add "<mount point>", "<map>" & Chr(11) & "<options>",
"<comment>"
'where:
'<mount point> - mount point name (key field)
'<map>         - the map file to consult for this mount point
'<options>     - mount options
'<comment>     - text comments
'NOTE The <map> and <options> are separated by a tab character,
'Chr(11), and form the value field for the entry.
map.add "/net", "-hosts" & Chr(11) & "-nosuid,nobrowse", "sample
mount"
'Open the NIS map named "auto.mount"
Set map = store.open("auto.mount")
'Add an entry to "auto.mount" NIS map. The input format is:
'map.add "<name>", "<Options>" & Chr(11) & "<network path>",
'"<comment>"
'where:
'<name>        - mount point name (key field)
'<options>     - mount options
'<network path> - the network path to consult for this mount
point
'<comment>     - text comments
'NOTE The <options> and <network path> are separated by a tab
'character, Chr(11), and form the value field for the entry.
map.add "cdrom", "-fstype=nsfs,ro" & Chr(11) & ":/dev/sr0",
"sample mount"
'Open the NIS map named "netgroup"
set map = store.open("netgroup")
'Add entries to the "netgroup" NIS map. The input format is:
'map.add "<group_name>", "<member_name>", "comment"
'where:
'<group_name> - defines the unique key of this group
'<member_name> - lists the members of the group. Each <member_
name>
'
'           is either another group name, all of whose
members
'
'           are to be included in the group being defined,
'
'           or a triple of the form:
'
'           (hostname,username,domainname)
'<comment>     - comments of this user
'Add a group "db_admin" with three members: dbas, dean, jon
map.add "db_admin", "dbas (,dean,) (firebird,jon,)", "All DBAs"
wScript.Echo "NIS map entries added."
...
```

## Commit

Commits the settings or changes for the map object to Active Directory.



## Syntax

```
void Commit();
```

## Discussion

The `Commit` method saves the settings of the map, but not the entries in the map.

## Exceptions

`Commit` throws an `ApplicationException` if access is denied.

## Example

The following code sample illustrates using `Commit` in a script:

```
SET cims = CreateObject("Centrify.DirectControl.Cims3")
SET zone = cims.GetZone("example.org/Zones/default")
SET store = CreateObject("Centrify.DirectControl.Nis.Store")
store.Attach zone.ADsPath, "username", "password"
SET map = store.Open("computers")
map.Name = "hosts"
map.Commit
```

## Exists

Checks whether there is an entry with a specific key-value pair.

## Syntax

```
bool Exists(string key, string value)
```

## Parameter

Specify the following parameters when using this method:

Parameter	Description
<code>key</code>	The key you want to check for.
<code>value</code>	The value you want to check for.

## Return value

Returns `true` if the specified entry exists.

• • • • •

## Get

Returns the map entry, or individual map record, with the specified key.

### Syntax

```
Entry Get(string key);
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
key	The key field that uniquely defines this entry in the NIS map.

### Return value

An entry object instance for the NIS map and key specified.

### Example

The following code sample illustrates using `Get` to retrieve an existing NIS map entry from a specific map.

```
...
'Specify the zone you want to work with
set zone = cims.GetZoneByPath("LDAP://CN=qa-
slovenia,CN=unix,DC=quantum,DC=net")
'Create the Store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and user credentials (username and
'password).
store.Attach zone.ADsPath, "nate.james", "my3w0rds"
'Open the NIS map named "Remote servers"
Set objMap = store.open("Remote servers")
'Specify the map entry key field
set objEntry = objMap.Get("helios")
wScript.Echo "IP: " & objEntry.Value
...
```

### GetByID

Returns the map entry, or individual map record, with the specified ID.



## Syntax

```
Entry GetByID(string id);
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
id	The ID that uniquely defines this entry in the NIS map.

## Return value

The entry object instance with the specified ID.

## GetDirectoryEntry

Returns the directory entry for the NIS map object.

## Syntax

```
DirectoryEntry GetDirectoryEntry ()
```

## Return value

The directory entry for the map object.

## Discussion

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

## GetEnumerator

Returns an enumeration of `IMap` objects.

## Syntax

```
IEnumerator GetEnumerator()
```



## Return value

The set of map entries.

## GetRedirectMap

Returns the redirect target NIS map.

## Syntax

```
Entry GetRedirectMap(Connection connection);
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>connection</code>	The Cims connection.

## Return value

The redirect target NIS map.

## Import

Imports a new map entry, or individual map record, with the specified key into Active Directory.

## Syntax

```
Entry Import(string key, string value, string comment);
```

## Parameters

Specify the following parameters when using this method.

Parameter	Description
<code>key</code>	The key field that uniquely defines this entry in the NIS map.
<code>value</code>	The value associated with the key field for this entry in the NIS map.
<code>comment</code>	Any optional text string to store in the comment field for this entry in the NIS map.

• • • • •

## Return value

An entry object for the map entry imported.

## Discussion

The difference between the `Import` and `Add` methods is that the `Import` method performs minimal checking and validation of data to maximize performance.

## Exceptions

`Import` throws an `ApplicationException` if the key or value is invalid.

## Example

The following code sample illustrates using `Import` to retrieve an existing NIS map entry from a specific map.

```
...
'Specify the zone you want to work with
set zone = cims.GetZoneByPath("LDAP://CN=qa-
slovenia,CN=unix,DC=quantum,DC=net")
'Create the Store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone.
'Provide the path to the zone and user credentials (username and
'password).
store.Attach zone.ADSPATH, "jae.smith", "pas$w0rd"
'Open the NIS map named "Remote servers"
Set map = store.open("Remote servers")
'Add an entry to the "Remote servers" NIS map. The input format
is:
map.import "oaxaca", "127.67.32.10", "Latin America Support
office"
...
```

## Remove

Removes the map entry with the specified key.

## Syntax

```
void Remove(string key);
```

## Parameter

Specify the following parameter when using this method:



---

Parameter	Description
key	The key field that uniquely defines this entry in the NIS map.

---

## Exceptions

Remove throws an `ApplicationException` if it can't find the `DirectoryEntry` value.

## Example

The following code sample illustrates using `Remove` to remove an existing NIS map entry by key name from a specific map:

```
...
'Identify the zone you want to work with
set zone = cims.GetZone("sample.com/centrify/zones/default")
'Create the store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and user credentials
store.Attach zone.ADsPath, "jae.smith", "pas$w0rd"
'Remove the "Modified_Key" map entry from the "generic map" NIS
map
set map = store.Open("generic map")
map.Remove "Modified_Key"
wScript.Echo "NIS map entry removed."
...
```

## RemoveByID

Removes the map entry with the specified ID.

## Syntax

```
void Remove(string id);
```

## Parameter

Specify the following parameter when using this method:

---

Parameter	Description
id	The ID that uniquely defines this entry in the NIS map.

---



## Exceptions

RemoveByID throws an `ApplicationException` if it can't find the `DirectoryEntry` object.

## IsReadable

Indicates whether the NIS map in the attached zone is readable.

## Syntax

```
bool IsReadable {get;}
```

## Property value

Returns `true` if the map object is readable by the user, or `false` if the map object is not readable.

## Discussion

This property returns a value of `true` if the user accessing the map entry object in Active Directory has sufficient permissions to read the entry properties.

## Example

The following code sample illustrates using this property in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath
("LDAP://CN=offshore,CN=unix,DC=quantum,DC=net")
'Create the Store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and username and password.
store.Attach objZone.ADsPath, "tae.parker", "9days^"
'Open the generic map type named "Workstations IDs"
Set map = store.open("workstations IDs")
'Check whether the map is readable
if not map.IsReadable then
    wScript.Echo "No read permission. Quitting application ..."
    wScript.Quit
end if
...
```

• • • • •

## IsWritable

Indicates whether the map object is writable.

### Syntax

```
bool IsWritable {get;}
```

### Property value

Returns true if the map object is writable by the user, or false if the map object is not writable.

### Discussion

This property returns a value of true if the user accessing the map object in Active Directory has sufficient permissions to change the map object's properties.

### Example

The following code sample illustrates using this property in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZoneByPath  
("LDAP://CN=pilot,CN=unix,DC=quantum,DC=net")  
'Create the Store object  
set store = CreateObject("Centrify.DirectControl.Nis.Store")  
'Attach to the target zone  
'Provide the path to the zone and username and password.  
store.Attach objZone.ADsPath, "jae.smith", "pas$wOrd"  
'Open the generic map type named "Workstations IDs"  
set map = store.open("workstations IDs")  
'Check whether the map is writable  
if not map.IsWritable then  
    wscript.Echo "No write permission for " & map.Name  
    wscript.Quit  
end if  
...
```

### Name

Gets or sets the map name.

• • • • •

## Syntax

```
string Name {get; set;}
```

## Property value

The map name.

## Discussion

You can specify any string for this property regardless of the type of NIS map.

## Exceptions

Name throws an `ArgumentException` if you try to set a value that is null, empty, or greater than 64 characters.

## Example

The following code sample illustrates using this property in a script:

```
...  
'Specify the zone you want to work with  
set objZone = cims.GetZoneByPath  
("LDAP://CN=pilot,CN=unix,DC=quantum,DC=net")  
'Create the Store object  
Set store = CreateObject("Centrify.DirectControl.Nis.Store")  
'Attach to the target zone  
'Provide the path to the zone and username and password.  
objStore.Attach objZone.ADsPath, "jae.smith", "pas$w0rd"  
'List map names  
For each map in objStore  
    wScript.Echo map.Name  
end if  
...
```

## Store

Gets the store object instance for the map object.

## Syntax

```
store store {get;}
```

## Property value

The Store instance for the map object.



## Type

Gets or sets the map type for the map object.

### Syntax

```
string Type {get; set;}
```

### Property value

The map type for the map object.

### Discussion

Internally, the map type defines how fields are parsed and interpreted for standard network maps and generic maps. The type value is used by Access Manager to identify the map type. Access Manager can recognize all of the common NIS map types.

### Example

The following code sample illustrates using this property in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath
("LDAP://CN=pilot,CN=unix,DC=quantum,DC=net")
'Create the Store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and username and password.
objStore.Attach objZone.ADsPath, "jae.smith", "pas$w0rd"
'Open the map type named "Workstations IDs"
Set objMap = objStore.Open("Workstations IDs")
'Check the map type
wscript.Echo "Map Type: " & objMap.Type
...
```

## Store

The Store class contains methods and properties used to manage a zone's NIS maps stored in Active Directory. This class is defined in the `Centrify.DirectControl.NISMap.API` namespace rather than the `Centrify.DirectControl.API` namespace.



## Syntax

```
public class IStore : ICloneable, IDisposable
```

## Discussion

The Store class supports the methods and properties that apply to all .NET objects. In addition to those methods and properties, the Store class provides some Centrify-specific methods and properties for managing network or generic NIS maps in a zone. Only Centrify-specific methods and properties are described in this reference.

## Methods

The Store class provides the following Centrify-specific methods:

This method	Does this
<b>Attach</b>	Links the Store object to the specified zone.
<b>Clone</b>	Makes a copy of the current Store object instance. Inherited from ICloneable.
<b>Create</b>	Creates a new NIS map object in the zone in Active Directory.
<b>Dispose</b>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from IDisposable.
<b>Delete</b>	Removes the specified NIS map from the zone and Active Directory.
<b>Exists</b>	Checks whether there is a NIS map with the specified name.
<b>GetDirectoryEntry</b>	Gets the directory entry for the NIS map container.
<b>Open</b>	Opens the NIS map with the specified name.

## Properties

The Store class provides the following Centrify-specific properties:

This property	Does this
<b>IsReadable</b>	Indicates whether the NIS map store is readable.
<b>IsWritable</b>	Indicates whether the NIS map store is writable.



## Attach

Attaches the `Centrify.DirectControl.NisMap.API` namespace storage object to the specified zone.

### Syntax

```
void Attach(string zonePath, string username, string password)  
void Attach(DirectoryEntry zoneEntry)
```

### Parameters

Specify the following parameters when using this method.

Parameter	Description
<code>zonePath</code>	The LDAP path to the zone to which you want to attach the NIS map storage object.
<code>username</code>	The user name to use when linking the NIS map storage object to the specified zone.
<code>password</code>	The user password to use when linking the NIS map storage object to the specified zone.
<code>zoneEntry</code>	The directory entry for the zone to which you want to attach the NIS map storage object. (.NET only)

### Exceptions

Attach may throw the following exception:

- `COMException` if an error occurs in a call to the underlying interface.
- `ApplicationException` if it fails to locate the NIS map store, the domain controller is read-only, access is not authorized, or an LDAP error occurs. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

### Example

The following code sample illustrates using `Attach` to create a map storage object and attach it to a zone:

```
...  
'Identify the zone in which the NIS maps will be created  
set zone = cims.GetZone("sample.com/centrify/zones/default")  
'Create the store object  
set store = CreateObject("Centrify.DirectControl.Nis.Store")
```



```
'Attach to the target zone.  
'Provide the path to the zone and user credentials (username and  
'password).  
store.Attach zone.ADSPATH, "jae.smith", "pas$w0rd"  
'Use store.Create to add a generic NIS map in this zone.  
store.Create "generic map","Generic Map"  
'Use store.Create to also add two auto_master NIS maps  
store.Create "auto.master","AutoMaster Map"  
store.Create "auto_master","AutoMaster Map"  
'Use store.Create to add a automount NIS map  
store.Create "auto.mount","Automount Map"  
'Use store.Create to add a netgroup NIS map  
store.Create "netgroup","Netgroup Map"  
wScript.Echo "NIS Maps added to " & zone.Name  
...
```

## Create

Creates a new NIS map with the specified name.

## Syntax

```
Map Create(string mapName, string type);
```

## Parameters

Specify the following parameters when using this method.

Parameter	Description
mapName	The name of the NIS map you want to create.
type	The type of NIS map to create.

## Return value

The map object created.

## Discussion

When you use this method to create NIS maps, you can specify just the map name or the map name and map type. Internally, however, the map name and type defines how fields are parsed and interpreted for standard network maps and generic maps. For example, the netgroup map name can only be used to create a netgroup type of NIS map. In most cases, you should only create generic maps (key value pairs) using this method to prevent NIS maps from becoming unusable due to unexpected formatting.



## Exceptions

Create may throw one of the following exceptions:

- `COMException` if there is an LDAP error. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.
- `ArgumentException` if the map name is not valid.

## Example

The following code sample illustrates using `Create` to add new NIS maps to a zone:

```
...
'Identify the zone in which the NIS maps will be created
set zone = cims.GetZone("sample.com/centrify/zones/default")
'Create the Store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone.
'Provide the path to the zone and user credentials (username and
'password).
store.Attach zone.ADsPath, "jae.smith", "pas$w0rd"
'Use store.Create to add a generic NIS map in this zone.
store.Create "Contact List","Generic Map"
'Use store.Create to also add the auto_master NIS maps
store.Create "auto_master","AutoMaster Map"
'Use store.Create to add a automount NIS map
store.Create "automounts","Automount Map"
'Use store.Create to add a netgroup NIS map
store.Create "netgroup","Netgroup Map"
wScript.Echo "NIS Maps added to " & zone.Name
...
```

## Delete

Deletes the specified NIS map.

## Syntax

```
void Delete(string mapName);
void Delete(IMap map);
```

## Parameter

Specify the following parameter when using this method:



---

Parameter	Description
mapName	The name of the NIS map you want to remove.
map	The NIS map you want to remove. (.NET only)

---

## Exceptions

Delete throws a `COMException` if there is an LDAP error. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using `Delete` to remove NIS maps from a zone:

```
...
'Identify the zone you want to work with
set zone = cims.GetZone("sample.com/centrify/zones/default")
'Create the Store object
set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and user credentials (username and
'password).
store.Attach zone.ADSPATH, "jae.smith", "pas$w0rd"
'Use store.Delete to delete the generic map named "generic map"
store.Delete "generic map"
wScript.Echo "NIS map deleted."
...
```

## Exists

Checks whether there is a NIS map with the specified name.

## Syntax

```
bool Exists(string mapName)
```

## Parameter

Specify the following parameter when using this method:

---

Parameter	Description
mapName	The map name whose existence you want to check for.

---



## Return value

Returns true if the specified NIS map exists.

## Exceptions

Exists throws a `COMException` if an error occurs during a call to the underlying interface.

## Example

The following code sample illustrates the use of `Store.Exists`:

```
SET cims = CreateObject("Centrify.DirectControl.Cims3")
  SET zone = cims.GetZone("example.org/Zones/default")
  SET store = CreateObject("Centrify.DirectControl.Nis.Store")
  store.Attach zone.ADsPath, "username", "password"
  IF NOT store.Exists("netgroup") THEN
    store.Create "netgroup", "408EE104-1864-41fa-B346-
19FED4092E68"
  END IF
```

## GetDirectoryEntry

Returns the directory entry for the NIS map container.

## Syntax

```
DirectoryEntry GetDirectoryEntry ()
```

## Return value

The `DirectoryEntry` attribute of the NIS map container.

## Discussion

This method can only be used in .NET programs because `DirectoryEntry` is a .NET-specific class for directory objects. This method cannot be used in COM-based programs.

## Open

Opens the NIS map with the specified name.



## Syntax

```
Map Open(string mapName);
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
mapName	The name of the NIS map you want to remove.

## Return value

The map object instance of the specified map, or null if the specified map is not found.

## Exceptions

Open throws a COMException if there is an LDAP error. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using Open to access a specific NIS map:

```
...  
'Identify the zone you want to work with  
set zone = cims.GetZone("sample.com/centrify/zones/default")  
'Create the store object  
Set store = CreateObject("Centrify.DirectControl.Nis.Store")  
'Attach to the target zone.  
'Provide the path to the zone and user credentials (username and  
'password).  
store.Attach zone.ADsPath, "jae.smith", "pas$w0rd"  
'Open the NIS map named "Remote servers"  
Set map = store.Open("Remote servers")  
...
```

## IsReadable

Indicates whether the NIS map storage object is readable.

• • • • •

## Syntax

```
bool IsReadable {get;}
```

## Property value

Returns `true` if the map storage object is readable by the user, or `false` if the map storage object is not readable.

## Discussion

This property returns a value of `true` if the user accessing the NIS map storage object in Active Directory has sufficient permissions to read its properties.

## Exceptions

`IsReadable` may throw one of the following exceptions:

- `ApplicationException` if the store entry cannot be located.
- `COMException` if there is an error in a call to the underlying interface.

## Example

The following code sample illustrates using this property in a script:

```
...
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath
("LDAP://CN=offshore,CN=unix,DC=quantum,DC=net")
'Create the Store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and username and password.
store.Attach objZone.ADsPath, "tae.parker", "9days^"
'Check whether the map node is readable
if not store.IsReadable then
    wscript.Echo "No read permission. Quitting application ..."
    wscript.Quit
end if
...
```

## IsWritable

Indicates whether the NIS map storage object is writable.

## Syntax

```
bool IsWritable {get;}
```



## Property value

Returns `true` if the map storage object is writable by the user, or `false` if the map storage object is not writable.

## Discussion

This property returns a value of `true` if the user accessing the NIS map storage object in Active Directory has sufficient permissions to change the storage object's properties.

## Exceptions

`IsWritable` throws a `COMException` if there is an LDAP error. LDAP errors can occur if the connection to the LDAP server fails, the connection times out, invalid credentials are presented, or there are other problems communicating with Active Directory.

## Example

The following code sample illustrates using this property in a script:

```
'Specify the zone you want to work with
set objZone = cims.GetZoneByPath
("LDAP://CN=offshore,CN=unix,DC=quantum,DC=net")
'Create the Store object
Set store = CreateObject("Centrify.DirectControl.Nis.Store")
'Attach to the target zone
'Provide the path to the zone and username and password.
store.Attach objZone.ADsPath, "tae.parker", "9days^"
'Check whether the map node is writable
if not store.IsWritable then
    wScript.Echo "No write permission. Quitting application ..."
    wScript.Quit
end if
...
```

## GroupInfo

The `GroupInfo` class contains methods and properties used to import and map UNIX group profiles to Active Directory groups. This class is defined in the `Centrify.DirectControl.API.Import` namespace.



## Syntax

```
public interface IGroupInfo : IDisposable
```

## Methods

The GroupInfo class provides the following methods:

This method	Does this
<b>Commit</b>	Commits changes to the pending group object and saves them in Active Directory.
<b>Delete</b>	Marks the pending group profile for deletion from Active Directory.
<b>Dispose</b>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from IDisposable.
<b>GetMembers</b>	Returns the members of a pending import group.
<b>Import</b>	Links the pending import group profile with the specified Active Directory group account.
<b>UpdateStatus</b>	Checks Active Directory for groups that match or conflict with a pending import group.

## Properties

The GroupInfo class provides the following properties:

This property	Does this
<b>CandidateDN</b>	Gets the distinguished name (DN) of the import candidate.
<b>GID</b>	Gets or sets the UNIX group identifier (GID) for the pending import group profile.
<b>ID</b>	Gets the unique ID of the pending import group object.
<b>IsImported</b>	Indicates whether the pending import group has been successfully imported.
<b>Members</b>	Gets all of a pending import group's members.
<b>Name</b>	Gets or sets the UNIX group name for a pending import group.
<b>Source</b>	Gets the text string that describes the source of the pending import data.
<b>Status</b>	Gets the status of the pending import group.



This property	Does this
<code>StatusDescription</code>	Gets a text string that provides detailed information about the status of the pending import group.
<code>TimeStamp</code>	Gets the date and time that the pending group profiles were imported from the data source.

## Commit

Commits any changes or updates to the pending group object and saves them in Active Directory.

### Syntax

```
void Commit()
```

## Delete

Marks the pending group profile object for deletion from Active Directory.

### Syntax

```
void Delete()
```

## Discussion

This method does not delete the pending group profile. After you mark the object for deletion, you must use the `Commit` method to commit changes to the object to Active Directory. When the `Commit` method is executed, the pending group profile is deleted from Active Directory to complete the operation.

## Exceptions

`Delete` throws an `UnauthorizedAccessException` if you have insufficient access rights to remove the UNIX profile in the zone.

## GetMembers

Returns the members of a pending import group.



## Syntax

```
string GetMembers()
```

## Return value

The collection of user profiles in the `UserInfos` object for the members of the pending import group.

## Discussion

This method returns the collection of user profiles that are members of the pending import group.

## Import

Imports the pending import group profile by associating the UNIX properties for the group with the specified Active Directory group account.

## Syntax

```
void Import(IGroup group)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>group</code>	The group for which you want to retrieve profile information.

## Discussion

This method links the pending import group to an Active Directory account and removes the group from the pending import list.

## UpdateStatus

Checks the Active Directory forest for matching or conflicting information that will allow or prevent a pending import group being imported.



## Syntax

```
void updateStatus()
```

## Discussion

This method searches Active Directory for a group name that matches the pending import group name and updates the pending import group properties with the results of the search. For example, if no Active Directory match is found or a UNIX profile already exists for the matching Active Directory group, the method updates the pending group's properties with that information.

**Note:** Checking the Active Directory forest for potential matching candidates or conflicts can be a time-intensive operation. Therefore, you should consider the size and distribution of the forest and limit the number of pending import groups you are working with when using this method.

## CandidateDN

Gets or sets the distinguished name (DN) of the import candidate.

## Syntax

```
string candidateDN {get; set;}
```

## Property value

The matching Active Directory group object for the pending group profile, if one is found. If there's no matching candidate in Active Directory, `null` is returned.

## Discussion

This property returns the Active Directory group account that appears to match the pending group profile. If there's an existing Active Directory group that matches the pending group, the pending import group can be mapped to that account. If no matching candidate is found in Active Directory, this property returns a `null` value.



## GID

Gets or sets the UNIX group identifier (GID) for the pending import group profile.

### Syntax

```
int GID {get; set;}  
long GID {get; set;}
```

### Property value

The UNIX group identifier (GID) for the pending group profile.

### Discussion

There are two versions of this property: one designed for COM-based programs that supports a 32-bit signed number one designed for .NET-based programs that allows a 64-bit signed number. Therefore, the data type for the property can be an integer (`int`) or a long integer (`long`) depending on the programming language you use.

## ID

Gets the unique ID of the pending import group object.

### Syntax

```
string ID {get;}
```

### Property value

The unique ID for the pending import group object.

## IsImported

Determines whether the pending import group has been successfully imported.

### Syntax

```
bool IsImported {get;}
```



## Property value

Returns `true` if the pending import group has been imported, or `false` if the group has not been successfully imported.

## Discussion

This property returns `true` if the pending import group has been imported, or `false` if the group has not been imported.

## Members

Gets all of a pending import group's members.

## Syntax

```
IGroupMembers Members {get;}
```

## Property value

The user names for the members of a pending import group.

## Name

Gets or sets the UNIX group name for a pending import group.

## Syntax

```
string Name {get; set;}
```

## Property value

The UNIX group name of a pending import group.

## Source

Gets the text string that describes the source of the pending import data.

## Syntax

```
string Source {get;}
```



## Property value

A text string that describes the source of the pending import data.

## Discussion

If the pending data was imported from a file, the property returns the source as `File` followed by the path to the file name imported. If the source of the pending import data was a NIS server, the property returns the NIS server name and domain. For example, if the source of the data was a file, the property returns a string similar to this:

```
File: C:\Migration\magnolia_groups
```

## Status

Gets the status of the pending import group.

## Syntax

```
statusType Status {get;}
```

## Property value

The status message for the pending import group.

## Discussion

The status is determined by checking Active Directory for existing groups that match or conflict with the pending import group. The property returns a number that determines the icon displayed for the group in the console. The icons indicate whether a group is:

When a group is	Status type	Icon displayed
Ready to import	Info	
Has potential issues that should be resolved	Warning	
Cannot be imported	Error	



## StatusDescription

Gets a text string that provides detailed information about the status of the pending import group.

### Syntax

```
string StatusDescription {get;}
```

### Property value

The status message for the pending import group.

### Discussion

The status is determined by checking Active Directory for existing groups that match or conflict with the pending import group. The results are displayed in Access manager and in the Status tab of a pending group's Properties dialog box. The status description can also include details about the members of the group. For example, if checking the Active Directory forest revealed a group name or GID conflict with an existing group or another pending import group, the StatusDescription property might include information similar to this:

```
There is another pending imported group using the same GID.  
There is another pending imported group using the same group  
name.  
Group member:'alan' cannot be associated.  
Group member:'rae' cannot be associated.
```

## TimeStamp

Gets the date and time that the pending group profiles were imported from the data source.

### Syntax

```
DateTime TimeStamp {get;}
```

### Property value

The date and time that the pending group data was imported.



## Example

The following code sample illustrates using this property in a script:

```
'Specify the zone you want to work with
Set objZone = cims.GetZone("w2k3.net/Centrify/Zones/default")
'Display the time groups where imported
Set objPendingGrps = objZone.GetImportPendingGroups
If not objPendingGrps is nothing then
    wScript.Echo "Imported from source: ",
objPendingGrps.TimeStamp
End if
...
```

## GroupInfos

The `GroupInfos` class contains methods and properties used to manage a collection of pending import group profiles. This class is defined in the `Centrifify.DirectControl.API.Import` namespace.

## Syntax

```
public interface IGroupInfos : IEnumerable<IGroupInfo>,
IDisposable
```

## Methods

The `GroupInfos` class provides the following methods:

This method	Does this
<code>Dispose</code>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.  Inherited from <code>IDisposable</code> .
<code>Find</code>	Returns the pending import group with the specified identifier from the collection of group profiles.
<code>GetEnumerator</code>	Returns an enumeration of <code>GroupInfo</code> objects.

## Properties

The `GroupInfos` class provides the following properties:



---

This property	Does this
<b>Count</b>	Determines the total number of pending import group profiles defined in the collection represented by the GroupInfos object.
<b>IsEmpty</b>	Determines whether the collection of pending import group profiles is empty.

---

## Find

Returns the pending import group with the specified identifier from the collection of group profiles.

### Syntax

```
IGroupInfo Find(string id)
```

### Parameter

Specify the following parameter when using this method:

---

Parameter	Description
<code>id</code>	The unique identifier of the pending group profile for which you want to retrieve information.

---

### Return value

The GroupInfo object for the specified pending import group.

## GetEnumerator

Returns an enumeration of IGroupInfo objects.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

The set of GroupInfo objects.



## Count

Determines the total number of pending import group profiles defined in the `GroupInfos` collection.

### Syntax

```
int Count {get;}
```

### Property value

The number of pending import group profiles in the set.

### Discussion

This property enumerates all of the profiles in the collection before it returns the `Count` value. If you only need to determine whether any import groups are pending, you should use the `IsEmpty` property for a faster response time.

## IsEmpty

Determines whether the collection of pending import group profiles is empty.

### Syntax

```
bool IsEmpty {get;}
```

### Property value

Returns `true` if there are no pending import group profiles in the `GroupInfos` object, or `false` if there is at least one pending import group profile in the object.

### Discussion

Unlike the `Count` property, the `IsEmpty` property does not enumerate all of the pending import profiles in the collection before it returns a value. If you only need to determine whether any profiles are defined, you should call this property for a faster response.



## GroupMember

The GroupMember class contains properties for working with the individual members of a pending import group. This class is defined in the `Centrify.DirectControl.API.Import` namespace.

### Syntax

```
public interface IGroupMember : IDisposable
```

### Methods

The GroupMember class provides the following method:

This method	Does this
<code>Dispose</code>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.
	Inherited from <code>IDisposable</code> .

### Properties

The GroupMember class provides the following properties:

This property	Does this
<code>CandidateDN</code>	Gets or sets the distinguished name (DN) of the pending import group members.
<code>Name</code>	Gets or sets the UNIX user name for a pending import group member.

### CandidateDN

Gets or sets the distinguished name (DN) of the pending import group members.

### Syntax

```
string CandidateDN {get; set;}
```



## Property value

The matching Active Directory group object for pending group profile, if one is found. If there's no matching candidate in Active Directory, nothing is returned.

## Discussion

This property returns the Active Directory user account that appears to match the pending group member. If there's an existing Active Directory user that matches the pending import group member, the pending import group member can be mapped to that account.

## Name

Gets or sets the UNIX user name for a pending import group member.

## Syntax

```
string Name {get; set;}
```

## Property value

The UNIX group name of a pending import group member.

## GroupMembers

The `GroupMembers` class contains properties used to manage a collection of pending import group members. This class is defined in the `Centrify.DirectControl.API.Import` namespace.

## Syntax

```
public interface IGroupMembers : IEnumerable<IGroupMember>,
IDisposable
```

## Methods

The `GroupMembers` class provides the following methods:



This method	Does this
<b>Add</b>	Adds a new UNIX user as a member of the pending import group.
<b>AddRange</b>	Adds a list of new UNIX users as members of the pending import group.
<b>Clear</b>	Removes all of the group members from a pending import group.
<b>Dispose</b>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from <code>IDisposable</code> .
<b>GetEnumerator</b>	Returns an enumeration of <code>GroupMember</code> objects. Inherited from <code>IEnumerable</code> .
<b>Remove</b>	Removes the specified group member from the list of members in a pending import group.

## Properties

The `GroupMembers` class provides the following properties:

This property	Does this
<b>Count</b>	Determines the total number of group members in the pending import group.

## Add

Adds a new UNIX user account as a member with the specified member name to the pending import group.

## Syntax

```
IGroupMember Add(string memberName)
```

## Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>memberName</code>	The UNIX user name of the account you want to add to the pending import group.

## Return value

The UNIX user you are adding as a member of the pending import group.



## AddRange

Adds a list of new UNIX user profiles as members of the pending import group.

### Syntax

```
void AddRange(ICollection string memberNames)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
memberNames	The list of UNIX user names you want to add as members of the pending import group.

## Clear

Removes all of the group members from a pending import group.

### Syntax

```
void Clear()
```

### Discussion

This method enables you to import a pending import group profile without resolving membership conflicts or mapping group members to Active Directory users.

## Remove

Removes the specified group member from the list of members in a pending import group.

### Syntax

```
void Remove(IGroupMember member)
```

### Parameter

Specify the following parameter when using this method:



---

Parameter	Description
member	The member you want to remove from the pending import group

---

## Count

Determines the total number of group members in the pending import group.

### Syntax

```
int Count {get;}
```

### Property value

The number of group members in the pending import group.

### Discussion

This property enumerates the list of members defined in the collection represented by the `GroupMembers` object.

## UserInfo

The `UserInfo` class contains methods and properties used to import and map UNIX user profiles to Active Directory user accounts. This class is defined in the `Centrify.DirectControl.API.Import` namespace.

### Syntax

```
public interface IUserInfo : IDisposable
```

### Methods

The `UserInfo` class provides the following methods:



This method	Does this
<b>Commit</b>	Commits any changes to the pending import user object and saves them in Active Directory.
<b>Delete</b>	Marks the pending user profile object for deletion from Active Directory.
<b>Dispose</b>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.  Inherited from <b>IDisposable</b> .
<b>GetCandidate</b>	Returns the directory object of a user pending import.
<b>Import</b>	Links the pending import user profile with the specified Active Directory user account.
<b>SetCandidate</b>	Sets the directory object of a user pending import.
<b>UpdateStatus</b>	Checks the Active Directory forest for matching or conflicting information that will allow or prevent a pending import user being imported.

## Properties

The `UserInfo` class provides the following properties:

This property	Does this
<b>CandidateDN</b>	Gets the distinguished name (DN) of the import candidate.
<b>Gecos</b>	Gets or sets the GECOS field of the UNIX profiles for the pending import user.
<b>HomeDirectory</b>	Gets or sets the home directory for the pending import user.
<b>ID</b>	Gets the unique ID of the pending import user object.
<b>Name</b>	Gets or sets the UNIX user name for a pending import user.
<b>PrimaryGroupID</b>	Gets or sets the UNIX group identifier (GID) of the primary group for the pending import user profile.
<b>Shell</b>	Gets or sets the default login shell for the pending import user.
<b>Source</b>	Gets the text string that describes the source of the pending import data.
<b>Status</b>	Gets the status of the pending import user.
<b>StatusDescription</b>	Gets a text string that provides detailed information about the status of the pending import user.
<b>TimeStamp</b>	Gets the date and time that the pending user profiles were imported from the data source.
<b>UID</b>	Gets or sets the UNIX user identifier (UID) for the pending import user profile.



## Commit

Commits any changes or updates to the pending import user object and saves them in Active Directory.

### Syntax

```
void Commit()
```

## Delete

Marks the pending user profile object for deletion from Active Directory.

### Syntax

```
void Delete()
```

### Discussion

This method does not delete the pending user profile. After you mark the object for deletion, you must use the **Commit** method to commit changes to the object to Active Directory. When the `Commit` method is executed, the pending user profile is deleted from Active Directory to complete the operation.

### Exceptions

`Delete` throws an `UnauthorizedAccessException` if you have insufficient access rights to remove the UNIX profile in the zone.

## GetCandidate

Returns the directory object of a user pending import.

### Syntax

```
DirectoryEntry GetCandidate()
```

### Return value

The directory object of a user that is a candidate for import. Returns `null` if the candidate cannot be found.



## Import

Imports the pending import user profile by associating the UNIX properties for the user with the specified Active Directory user account.

### Syntax

```
void Import(IUser user)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
user	The user for which you want to retrieve profile information.

### Discussion

This method links the pending import user to an Active Directory account and removes the user from the pending import list.

## SetCandidate

Sets the directory object of a user pending import.

### Syntax

```
void SetCandidate(DirectoryEntry entry);
```

### Parameters

Specify the following parameter when using this method.

Parameter	Description
entry	The directory entry for the user that is a candidate for import.

### Discussion

This method updates the **CandidateDN** property,



## UpdateStatus

Checks the Active Directory forest for matching or conflicting information that will allow or prevent a pending import user being imported.

### Syntax

```
void UpdateStatus()
```

### Discussion

This method searches Active Directory for a user name that matches the pending import user name and updates the pending import user properties with the results of the search. For example, if no Active Directory match is found or a UNIX profile already exists for the matching Active Directory user, the method updates the pending user's properties with that information.

**Note:** Checking the Active Directory forest for potential matching candidates or conflicts can be a time-intensive operation. Therefore, you should consider the size and distribution of the forest and limit the number of pending import users you are working with when using this method.

## CandidateDN

Gets or sets the distinguished name (DN) of the import candidate.

### Syntax

```
string CandidateDN {get; set;}
```

### Property value

The matching Active Directory user object for pending user profile, if one is found. If there's no matching candidate in Active Directory, nothing is returned.

### Discussion

This property returns the Active Directory user account that appears to match the pending user profile. If there's an existing Active Directory user that matches the pending user, the pending import user can be mapped to that



account. If no matching candidate is found in Active Directory, this property returns a null value.

## **Gecos**

Gets or sets the GECOS field of the UNIX profile for the pending import user.

### **Syntax**

```
string Gecos {get; set;}
```

### **Property value**

The text string value of the GECOS field in the UNIX profile for the pending import user.

## **HomeDirectory**

Gets or sets the home directory field of the UNIX profile for the pending import user.

### **Syntax**

```
string HomeDirectory {get; set;}
```

### **Property value**

The text string value of the home directory field in the UNIX profile for the pending import user.

## **ID**

Gets the unique ID of the pending import user object.

### **Syntax**

```
string ID {get;}
```

### **Property value**

The unique ID for the pending import group object.

• • • • •

## Name

Gets or sets the UNIX user name for a pending import user.

### Syntax

```
string Name {get; set;}
```

### Property value

The UNIX user name of a pending import user.

## PrimaryGroupID

Gets or sets the UNIX group identifier (GID) of the primary group for the pending import user profile.

### Syntax

```
int PrimaryGroupID {get; set;}
```

### Property value

The UNIX group identifier (GID) of the primary group for the pending import user profile.

## Discussion

There are two versions of this property: one designed for COM-based programs that supports a 32-bit signed number one designed for .NET-based programs that allows a 64-bit signed number. Therefore, the data type for the property can be an integer (`int`) or a long integer (`long`) depending on the programming language you use.

## Shell

Gets or sets the default login shell for the pending import user.

### Syntax

```
string shell {get; set;}
```



## Property value

The text string value of the default login shell in the UNIX profile for the pending import user.

## Source

Gets the text string that describes the source of the pending import data.

## Syntax

```
string source {get;}
```

## Property value

A text string that describes the source of the pending import data.

## Discussion

If the pending data was imported from a file, the property returns the source as `File:` followed by the path to the file name imported. If the source of the pending import data was a NIS server, the property returns the NIS server name and domain. For example, if the source of the data was a file, the property returns a string similar to this:

```
File: C:\Migration\magnolia_passwd
```

## Status

Gets the status of the pending import user.

## Syntax

```
statusType Status {get;}
```

## Property value

The status message for the pending import user.



## Discussion

The status is determined by checking Active Directory for existing users that match or conflict with the pending import user. The property returns a number that determines the icon displayed for the user in the console. The icons indicate whether a pending user is:

When a user is	Status type	Icon displayed
Ready to import	Info	
Has potential issues that should be resolved	Warning	
Cannot be imported	Error	

## StatusDescription

Gets a text string that provides detailed information about the status of the pending import user.

## Syntax

```
string StatusDescription {get;}
```

## Property value

The detailed status message for the pending import user.

## Discussion

The status is determined by checking Active Directory for existing users that match or conflict with the pending import user. The results are displayed in Access Manager and in the Status tab of a pending user's Properties dialog box. The status description can also include details about the user's primary group. For example, if checking the Active Directory forest revealed a user name or UID conflict with an existing user or another pending import user, the `StatusDescription` property might include information similar to this:

```
No group with the corresponding GID found in Active Directory.  
There is another pending imported user using the same UID.  
There is another pending imported user using the same user name.
```

• • • • •

## TimeStamp

Gets the date and time that the pending user profiles were imported from the data source.

### Syntax

```
DateTime TimeStamp {get;}
```

### Property value

The date and time that the pending user data was imported.

### Example

The following code sample illustrates using this property in a script:

```
'Specify the zone you want to work with
Set objZone = cims.GetZone("w2k3.net/Centrify/Zones/default")
'Display the time users where imported
Set objPendUsers = objZone.GetImportPendingUsers
If not objPendUsers is nothing then
    wScript.Echo "Imported from source: ", objPendUsers.TimeStamp
End if
...
```

## UID

Gets or sets the UNIX user identifier (UID) for the pending import user profile.

### Syntax

```
int UID {get; set;}
```

### Property value

The UNIX user identifier (UID) for the pending import user profile.

### Discussion

There are two versions of this property: one designed for COM-based programs that supports a 32-bit signed number one designed for .NET-based programs that allows a 64-bit signed number. Therefore, the data type for the property



can be an integer (`int`) or a long integer (`long`) depending on the programming language you use.

## UserInfos

The `UserInfos` class contains methods and properties used to manage a collection of pending import user profiles. This class is defined in the `Centrify.DirectControl.API.Import` namespace.

### Syntax

```
public interface IUserInfos : IDisposable
```

### Methods

The `UserInfos` class provides the following methods:

This method	Does this
<code>Dispose</code>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from <code>IDisposable</code> .
<code>Find</code>	Returns the pending import user with the specified identifier from the collection of pending user profiles.
<code>GetEnumerator</code>	Returns an enumeration of <code>UserInfo</code> objects.

### Properties

The `UserInfos` class provides the following properties:

This property	Does this
<code>Count</code>	Gets the total number of pending import user profiles defined in the collection represented by the <code>UserInfos</code> object.
<code>IsEmpty</code>	Indicates whether the collection of pending import users is empty.



## Find

Returns the pending import user with the specified identifier from the collection of pending user profiles.

### Syntax

```
IUserInfo Find(string id)
```

### Parameter

Specify the following parameter when using this method:

Parameter	Description
id	The unique identifier of the pending user profile for which you want to retrieve information.

### Return value

The UserInfo object for the specified pending import user.

## GetEnumerator

Returns an enumeration of UserInfo objects.

### Syntax

```
IEnumerator GetEnumerator()
```

### Return value

The set of UserInfo objects.

## Count

Gets the total number of pending import user profiles defined in the collection represented by the UserInfos object.

### Syntax

```
int Count {get;}
```



## Property value

The number of pending import user profiles in the set.

## Discussion

This property enumerates all of the profiles in the collection before it returns the `Count` value. If you only need to determine whether any pending import groups, you should use the `IsEmpty` property for a faster response time.

## IsEmpty

Determines whether the collection of pending import user profiles is empty.

## Syntax

```
bool IsEmpty {get;}
```

## Property value

Returns `true` if there are no pending import user profiles in the `UserInfos` object, or `false` if there is at least one pending import user profile in the object.

## Discussion

If there are no pending import user profiles in the `UserInfos` object, this property returns `true`. If there is at least one pending import user profile, the property returns `false`. Unlike the `Count` property, the `IsEmpty` property does not enumerate all of the pending import profiles in the collection before it returns a value. If you only need to determine whether any profiles are defined, you should call this property for a faster response.



# Data storage for Centrify zones

This chapter provides a detailed description of how Centrify-specific information is stored in Active Directory. Understanding how this information is stored will enable you to manipulate data for UNIX users and groups using standard LDAP tools, such as `ldapadd`, directly from UNIX computers, or using Active Directory LDAP utilities, such as `adinfo`, on Windows computers without using Access Manager, COM objects, or .NET programs.

## Basic requirements

Centrify stores UNIX-specific properties for users, groups, and computers, as well as zones and zone properties, within Active Directory by adhering to Microsoft standards for data storage. Because of this adherence to Microsoft standards, Centrify stores the UNIX-specific information differently depending on the Active Directory schema you are using and the type of Centrify zones you create.

The Centrify Windows API provides a logical abstraction of the data model so that you can manipulate Centrify-specific information without understanding the differences between zone types. If you want to manipulate the data directly without the logical abstraction, however, you need to understand the details of how UNIX-specific properties and zone information are stored for each type of zone and schema. Once you have a more detailed understanding of the physical and logical data model for each zone type, you may be able to perform tasks that are not possible with the Access Manager console.



## Schemas and zones

Centrify stores UNIX identity data and Centrify zone data in Active Directory, without modifying or extending the standard Active Directory schema. Centrify stores UNIX account profiles in standard text properties in an existing Active Directory object. This data model can be used with the default Active Directory schema or with any standard schema extension provided by Microsoft. Zones and their properties are stored in the same manner, using standard text properties in an existing Active Directory object.

The default data storage model and Centrify zones enable a single Active Directory user account to be associated with any number of unique UNIX profiles that a user may have across your environment. These UNIX profiles can have unique UIDs, GIDs, home directories, and preferred shells on one or more different UNIX systems.

### Supported zone types

Standard Centrify zones use the default data storage model. However, Centrify can also support the RFC 2307 data model if you are using the Microsoft RFC 2307 schema extension, or the Microsoft Services for UNIX (SFU) data model if you are using that schema extension. To support these schema extensions, you can choose the zone type you want to use for each zone.

The following table lists the supported zone types and the relationship between the zone type and the Active Directory schema.

Zone type	Active Directory schema
Classic Centrify zones, versions 2.x, 3.x, and 4.x	Any schema
Hierarchical standard zones, version 5.x or later	Any schema
Classic RFC 2307-compatible zones, versions 2.x, 3.x, and 4.x	RFC 2307-compliant schema
Hierarchical RFC 2307-compatible zones, version 5.x or later	RFC 2307-compliant schema
SFU-compatible zones, version 3.5	Microsoft Services for UNIX (SFU), version 3.5
SFU-compatible zones, version 4.0	Windows Services for UNIX (SFU), version 4.00

**Note:** Classic RFC 2307-compatible zones require Active Directory Dynamic Auxiliary Classes. The forest functional level must be at



least Windows Server 2003 to use Dynamic Auxiliary Classes. All hierarchical zones require the domain functional level to be at least Windows Server 2003.

## How the zone type can affect features

Because the details of the data model depend on the Active Directory schema and the zone type, the zone type can also impact the features a particular zone can support. For example, classic and hierarchical Centrify zones support multiple profiles for each user but SFU zones do not. The following table summarizes key differences between zone types.

Zone type	Multiple profiles	Delegation	Inheritance
Classic Centrify zones (2.x, 3.x, 4.x)	Yes	Yes	No
Classic RFC 2307-compatible	Yes	Yes	No
Hierarchical Centrify zones (5.x)	Yes	Yes	Yes
Hierarchical RFC 2307-compatible zones	Yes	Yes	Yes
Hierarchical Services for UNIX (SFU) zones	No	No	Yes*
Services for UNIX (SFU), version 3.5, zones	No	No	No
Services for UNIX (SFU), version 4.0, zones	No	No	No

\* A hierarchical SFU zone can only be the root parent zone. You cannot create any Service for UNIX zone as a child zone.

For more information about differences in how data is stored in a specific zone type, see [Differences between types of zones](#).

## The logical data model for objects

This section describes the logical data model associated with each type of Centrify object. The data types used in the discussion of the logical data model, however, may not reflect the actual implementation of the data for a given zone type. For example, a user's UID value might be stored as an integer in SFU



zones or as a string in Centrify zones, but represented as an integer (int) in the logical data model.

Similarly, the names used in the logical data model may not reflect the actual Active Directory attribute names for a given zone type. For example, in Centrify zones, there are UNIX-specific attributes, such as the UID value, that are stored in an Active Directory object where the schema does not have a corresponding attribute, whereas the schema for SFU zones provides this attribute.

## Use of existing attributes

Centrify uses existing Active Directory attributes to store data. For example, most Centrify zones use Active Directory `serviceConnectionPoint` objects to store UNIX-specific data. The `serviceConnectionPoint` class is intended to hold information about services. The `keywords` attribute of the `serviceConnectionPoint` object holds name-value pairs that an Active Directory service can use to store its own attributes.

For example, if you were to use `ldapsearch` to filter the `keywords` attribute for a user's `serviceConnectionPoint` class in a Centrify zone, you would see results similar to the following:

```
keywords: foreign:False
keywords: gid:800
keywords: home:/home/jae
keywords: parentLink:S-1-5-21-3619765212-102450798-26543
keywords: shell:/bin/bash
keywords: uid:810
keywords: unixEnabled:True
```

Once you are familiar with the logical data model for Centrify objects, refer to the appropriate zone-specific section for more detailed information about which Active Directory attributes are used to store data in a particular type of zone.

## Logical data attributes for zones

The following table describes the logical attributes for Centrify zone objects.

Logical attribute	Type	Description
<code>NextUid</code>	int	The value of the next UID that will be allocated automatically.
<code>NextGid</code>	int	The value of the next GID that will be allocated



Logical attribute	Type	Description
		automatically.
ReservedUids	string	UIDs that should not be used in the automatic allocation sequence.
ReservedGids	string	GIDs that should not be used in the automatic allocation sequence.
AvailableShells	string	Which shells are available in the zone. This attribute is used to populate the list of available shells that can be assigned in the user's UNIX profile.
DefaultHomeDirectory	string	The default value for the user's home directory. Typically, this attribute contains the string <code>\${user}</code> which is replaced with the user's login name in the home directory path. For example, if this attribute is <code>/home/\${user}</code> , and the user's login name is <code>shea</code> , the user's home directory path is defined as <code>/home/shea</code> .
DefaultShell	string	The default shell to use for the zone.
ZoneName	string	The name of the zone.
DefaultGid	int	The value of the GID for the group profile associated with the Active Directory group used as the default group for new users.
Description	string	The text string to use as a description for the zone. This attribute is used to display additional information about a zone in Access Manager. For example, if the value for <b>Description</b> attribute is <b>Business development zone</b> , the console displays this string after the zone name:

## Logical data attributes for users

The following table describes the logical attributes for the UNIX user object. Most of these attributes represent elements of the user's entry in the `/etc/passwd` file. The last two are specific to Centrify.

Logical attribute	Type	Description
Uid	int	The value of the user's UID in the UNIX profile.
UnixName	string	The user's login name in the UNIX profile.
Gid	int	The value of the user's primary group identifier (GID).
Home	string	The path to use for the user's home directory in the user's UNIX profile.
Shell	string	The path to use for the user's shell in the user's UNIX profile.
UnixEnabled	bool	Whether the user is allowed to log on to computers in classic zones.



Logical attribute	Type	Description
		This attribute allows a user profile to exist in a classic zone but not allowed to log on to any computers. Disabling user access is particularly useful for recording who used to own a retired UID. This attribute is not used in hierarchical zones.
AppEnabled	bool	Not used. <b>Note</b> You might see this attribute if you have a legacy version of Centrify software installed in your environment.

## Logical data attributes for groups

The following table describes the logical attributes for the UNIX group object. Note that there is no attribute for group membership. Group membership for groups with a UNIX profile is always derived from the Active Directory group membership.

Logical attribute	Type	Description
Gid	int	The value of the group identifier (GID) for the group's UNIX profile.
UnixName	string	The group name for the group's UNIX profile.

## Logical data attributes for computers

The following table describes the logical attributes for the UNIX computer object. The computer attributes are used to provide specific details about computers joined to a zone. In most cases, you should not change these attributes. They are created automatically by the `adjoin` process.

Logical attribute	Type	Description
cpuCount	int	The number of CPUs detected on a computer joined to the domain. This attribute was used for licensing calculations in earlier versions of Centrify software (versions 3.0.x through 4.1.x).
agentVersion	string	The version of the Centrify agent installed on a computer joined to the domain.



## Logical data attributes for NIS maps

The NIS maps for a zone are stored in a `NisMaps` container object. The `NisMaps` container object is similar to the `Users`, `Groups`, or `Computers` container objects that contains a zone's UNIX data. Each individual NIS map object is also a container object. The name of the object and its GUID are its only attributes. Because the NIS map objects don't require any special mapping between a logical attribute name and an Active Directory attribute name, the standard Active Directory attribute names are used. The following table describes the Active Directory attributes for NIS map objects.

Logical attribute	Type	Description
Common Name (cn)	string	The name of the NIS map.
Description	string	The GUID of the map type. This attribute is used by the Access Manager console. If the attribute value is not specified, the console attempts to resolve to the most appropriate map type.

Under each NIS map object, each map entry is stored as a key/value pair of `classStore` objects. The following table describes the Active Directory attributes for the entry objects.

Logical attribute	Type	Description
Common Name (cn)	string	The unique key of the key/value pair.
Description	string	The key for the map entry.
adminDescription	string	The value for the map entry.
wwwHomePage	string	The text comment for the map entry. This attribute is only used to display the comment in Access Manager. The comment cannot be served to NIS clients.

## Differences between types of zones

The user and group attributes described in the logical data model are stored differently in Centrify zones than they are in SFU zones.

- When you have the Microsoft Services for UNIX (SFU) schema extension, version 3.5 or version 4.0, and use SFU-compatible zones, user and group UNIX attributes are stored in the Active Directory user and Active Directory group objects.

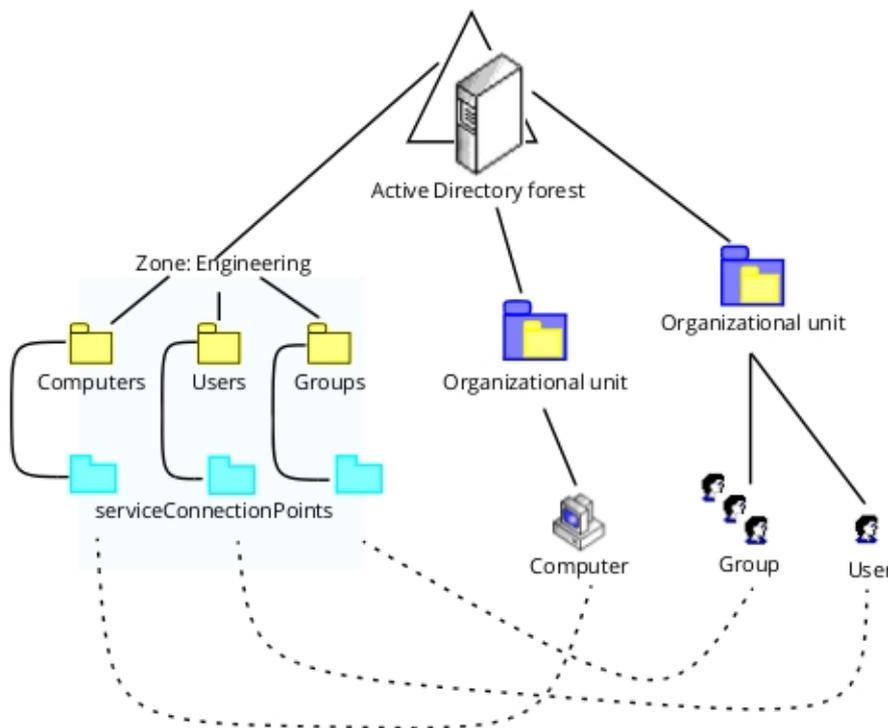


- In classic and hierarchical Centrify and RFC 2307-compatible zones, user and group UNIX attributes are stored in one `serviceConnectionPoint` object per zone for each user and group.

## Classic Centrify zones (2.x, 3.x, 4.x)

In classic Centrify zones, each zone is a separate tree stored in the directory. The root of the zone tree is an Active Directory container with the same name as the zone. The zone attributes described in the logical data model are stored in the attributes of this container object. Within the zone container, there are sub-containers for the **Users**, **Groups**, and **Computers** in the zone.

The following figure illustrates the basic structure used for classic zones.



Within each of the sub-containers, there are `serviceConnectionPoint` (SCP) objects. The `serviceConnectionPoint` (SCP) objects contain the Centrify attributes for each user, group, or computer defined for the zone. Each of user, group, or computer `serviceConnectionPoint` objects also has a link back to its parent object (shown as dotted lines in the figure above).

**Note:** Although Figure 1 illustrates the basic layout for a classic zone using a simple scenario, more complex configurations are possible. For example, in the illustrated scenario, the parent user

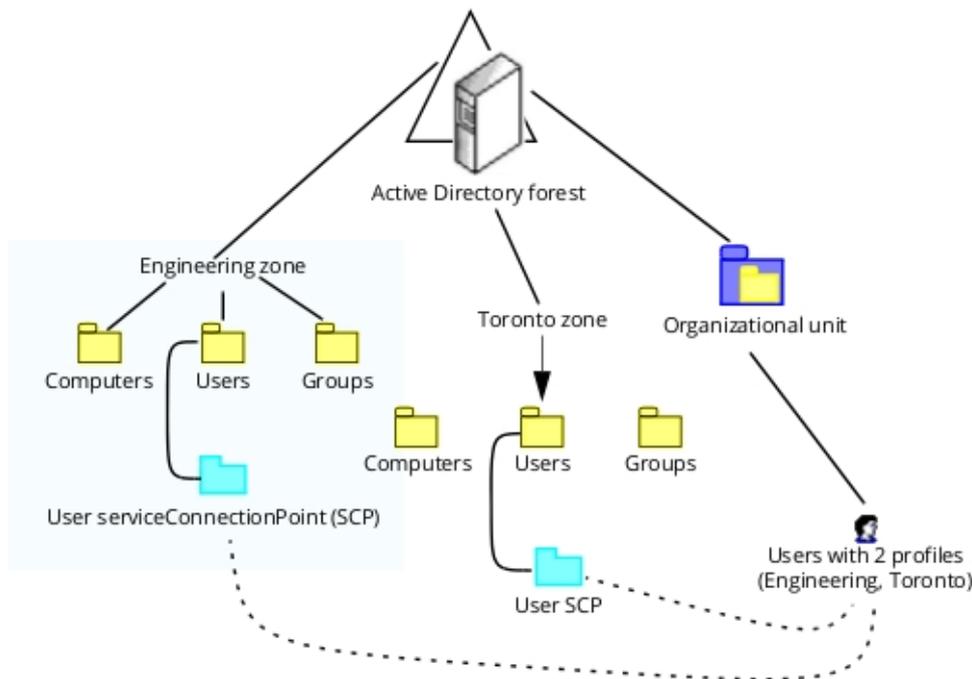


and group objects are in the same organizational unit (OU), but this is not a requirement. Similarly, the zone tree does not need to be in the same domain as the user or computers objects.

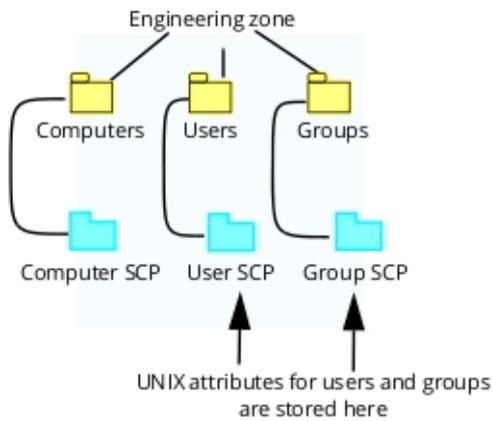
The zone tree structure separates Centrify and UNIX-specific attributes for each zone from every other zone and from the base Active Directory objects for the users and groups. This structure has the following important benefits:

- It enables a single Active Directory user to have many different UNIX profiles.
- It enables you to delegate administrative tasks to users and groups on a zone-by-zone basis.

The following figure illustrates how the zone tree structure enables a single Active Directory user to have many different UNIX profiles.



In a classic zone, the Centrify and UNIX-specific attributes are separate from all of the other zones and from the base Active Directory objects for the users and groups. This enables delegated management of UNIX-related tasks, such as adding or removing UNIX profiles, within each zone.



## Parent link attributes

For each `serviceConnectionPoint` object in the zone, there is a link back to the corresponding Active Directory object that owns the `serviceConnectionPoint` object. This link is stored in one of two ways:

- Using the `ParentLink` pseudo-attribute. This attribute contains the string security identifier (SID) of the parent Active Directory object. This attribute is used in Centrify, version 3.x and later.
- Using the `managedBy` Active Directory attribute. This attribute is set to the distinguished name (DN) of the parent object. This attribute was used in Centrify, version 2.x, but discontinued because there are cases when it is not possible to set the `managedBy` attribute.

## Zone attributes in classic Centrify zones

The zone object class is stored as a container object. The common name (cn) of the object must be set to the zone name. Most of the other attributes for a zone are stored as pseudo-attributes using the `Active Directory description` attribute. The following table summarizes how zone attributes are stored in Active Directory for Centrify zones.

Zone attribute	Stored in Active Directory attribute
	<code>cn:ZoneName</code>
<code>ZoneName</code>	For example: <code>cn:default</code>
<code>ZoneVersion</code>	<code>displayName:ZoneVersion</code>



Zone attribute	Stored in Active Directory attribute
	<p>This attribute determines compatibility between a zone object and the Access Manager console. The valid values are:</p> <ul style="list-style-type: none"><li>■ <code>\$CimsZoneVersion2</code> for zones compatible with Centrify versions 2.x and 3.x</li><li>■ <code>\$CimsZoneVersion3</code> for RFC 2307 classic zones compatible with Centrify versions 2.x and 3.x</li><li>■ <code>\$CimsZoneVersion4</code> for classic zones compatible with Centrify, version 4.x</li><li>■ <code>\$CimsZoneVersion5</code> for hierarchical zones</li></ul> <p>For example:</p> <pre>displayName: \$CimsZoneVersion5</pre>
Description	<pre>description:description:value</pre> <p>For example:</p> <pre>description:description:Pilot EMEA</pre>
NextUid	<pre>description:uidnext:value</pre> <p>For example:</p> <pre>description:uidnext:12098</pre>
NextGid	<pre>description:gidnext:value</pre> <p>For example:</p> <pre>description:gidnext:12098</pre>
ReservedUids	<pre>description:uidreserved:value</pre> <p>This attribute can be a multi-valued list, using a colon as the separator. Values can be individual numbers or a range of numbers separated with a dash character (-).</p> <p>For example:</p> <pre>description:uidreserved:0-99:501</pre>
ReservedGids	<pre>description:gidreserved:value</pre> <p>This attribute has the same format as the <code>reserveduids</code> attribute. For example:</p> <pre>description:gidreserved:1000-2500</pre>
Availableshells	<pre>description:availableshells:value</pre> <p>This attribute can be a multi-valued list of shell names, using a colon as the separator.</p> <p>For example:</p> <pre>description:availableshells:/bin/sh</pre>
DefaultHomeDirectory	<pre>description:defaulthome:value</pre>



Zone attribute	Stored in Active Directory attribute
	For example: description:defaultHome:/nfs/\${user}
DefaultShell	For example: description:defaultShell:/bin/bash
DefaultGroup	For example: description:defaultgid:12098 schema:Dynamic_Schema_Version
ZoneType	<p>This attribute identifies the schema layout a zone object uses. The valid values are:</p> <ul style="list-style-type: none"><li>■ <b>Dynamic_Schema_1_0</b> for Centrify, version 1.0, zones. This schema type is obsolete for version 2.x and later.</li><li>■ <b>Dynamic_Schema_2_0</b> for classic Centrify zones, 2.x and 3.x compatible.</li><li>■ <b>Dynamic_Schema_3_0</b> for classic Centrify zones, 3.x and 4.x compatible.</li><li>■ <b>Dynamic_Schema_5_0</b> for hierarchical Centrify zones, 5.x compatible.</li><li>■ <b>SFU_3_0</b> for SFU zones with the Microsoft Services for UNIX (SFU), version 3.x, schema extension.</li><li>■ <b>SFU_4_0</b> for SFU zones with the Microsoft Services for UNIX (SFU), version 4.x, schema extension.</li><li>■ <b>CDC_RFC_2307</b> for classic RFC 2307-compliant zones, Centrify 2.x and 3.x compatible.</li><li>■ <b>CDC_RFC_2307_2</b> for classic RFC 2307-compliant zones, Centrify 4.x compatible.</li><li>■ <b>CDC_RFC_2307_3</b> for hierarchical RFC 2307-compliant zones, Centrify 5.x compatible.</li></ul> <p>For example: description:schema:Dynamic_Schema_5_0</p>

## User attributes in classic Centrify zones

A user extension object is a `serviceConnectionPoint` object that is created in the `users` sub-container of the zone. The pseudo-attributes for this object are stored in the `keywords` attribute.



User attribute	Stored in Active Directory attribute
UnixName	<p>cn:userlogin</p> <p>For SCP objects, the <b>Name</b> attribute is a logical pointer that is the same as the CN attribute. You can use either attribute to store the user's UNIX login name.</p> <p>For example:</p> <p>cn:cain</p>
UserVersion	<p>displayName:UserVersion</p> <p>This attribute determines compatibility between a user profile object and the Access Manager console. The only valid value for this attribute is <b>\$CimsUserVersion2</b>.</p> <p>For example:</p> <p>displayName:\$CimsUserVersion2</p>
ParentLink	<p>managedBy:DN_ActiveDirectoryUser</p> <p>You can use the <b>managedBy</b> or <b>keywords</b> attribute to store the <b>parentLink</b>. If the zone is a 2.x and 3.x compatible zone, you should set this attribute to the DN of the parent Active Directory user object.</p> <p>For example:</p> <p>managedBy:cn=ben.lau,cn=users,dc=ice,dc=net</p> <p>If the zone does not need to be compatible with older versions of Centrify software, you can use the <b>keywords</b> attribute and <b>parentLink</b> pseudo-attribute to specify the security identifier (SID) of the parent Active Directory user object.</p> <p>For example:</p> <p>keywords:parentLink:S-n-n-nn-nnn..</p>
Uid	<p>keywords:uid:value</p> <p>For example:</p> <p>keywords:uid:458</p>
Gid	<p>keywords:gid:value</p> <p>For example:</p> <p>keywords:gid:458</p>
Home	<p>keywords:home:value</p> <p>For example:</p> <p>keywords:home:/home/shear</p>
Shell	<p>keywords:shell:value</p> <p>For example:</p> <p>keywords:shell:/bin/bash</p>



User attribute	Stored in Active Directory attribute
	keywords:unix_enabled:value
UnixEnabled	For example: keywords:unix_enabled:False
	keywords:foreign:value
ForeignForest	This attribute indicates whether a user in a zone is from an external forest. For example: keywords:foreign:False
AppEnabled	This attribute is no longer used.

## Group attributes in classic Centrify zones

A group extension object is a `serviceConnectionPoint` object that is created in the `Groups` sub-container of the zone. The pseudo-attributes for this object are stored in the `keywords` attribute.

Group attribute	Stored in Active Directory attribute
	name:GroupName
UnixName	For SCP objects, the <code>Name</code> attribute is the same as the <code>CN</code> attribute. Either attribute can be set, but attribute use should be consistent with other objects. For example: name:performx
	displayName:GroupVersion
GroupVersion	This attribute determines compatibility between a group profile object and the Access manager console. The only valid value for this attribute is <code>\$CimsGroupVersion3</code> . For example: displayName:\$CimsGroupVersion3
	managedBy:DN_ActiveDirectoryGroup
ParentLink	If the zone is a 2.x and 3.x compatible zone, you should set this attribute to the DN of the parent Active Directory group object. For example: managedBy: cn=interns,cn=users,dc=ice,dc=net  If the zone does not need to be compatible with older versions of Centrify software, you can use the <code>keywords</code> attribute and <code>parentLink</code> pseudo-attribute to specify the security identifier (SID) of the parent Active Directory group object.



Group attribute	Stored in Active Directory attribute
	For example: <code>keywords:parentLink:S-n-n-nn-nnn..</code>
Gid	For example: <code>keywords:gid:458</code>
UnixEnabled	This attribute is only applicable in classic 4.x zones.
ForeignForest	Not supported in 3.x or 4.x.

## Computer attributes in classic Centrifify zones

A computer extension object is a `serviceConnectionPoint` object that is created in the `Computers` sub-container of the zone. The pseudo-attributes for this object are stored in the `keywords` attribute.

Computer attribute	Stored in Active Directory attribute
	<code>name:ComputerName</code>
	For SCP objects, the <b>Name</b> attribute is the same as the <b>CN</b> attribute. Either attribute can be set, but attribute use should be consistent with other objects.
UnixName	Normally, computer attribute values are set by the <code>adjoin</code> process and do not need to be modified.  For example: <code>name:magnolia.ajax.org</code>
	<code>displayName:ComputerVersion</code>
ComputerVersion	This attribute determines compatibility between a computer profile object and the Access Manager console. The only valid value for this attribute is <code>\$CimsComputerVersion3</code> .  For example: <code>displayName:\$CimsComputerVersion3</code>
	<code>managedBy:DN_ActiveDirectoryGroup</code>
	If the zone is a 2.x and 3.x compatible zone, you should set this attribute to the DN of the parent Active Directory computer object.
ParentLink	For example: <code>managedBy:cn=hr,cn=computers,dc=ajax,dc=org</code>  If the zone does not need to be compatible with older versions of Centrifify software, you can use the <code>keywords</code> attribute and <code>parentLink</code> pseudo-attribute to specify the security identifier (SID) of

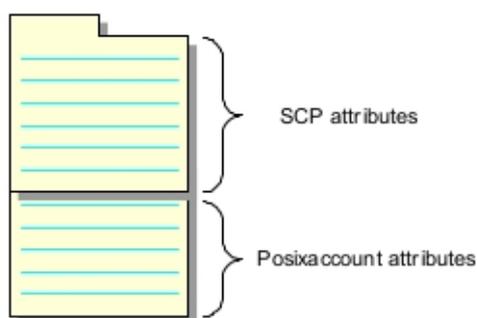


Computer attribute	Stored in Active Directory attribute
	<p>the parent Active Directory computer object.</p> <p>For example:</p> <pre>keywords:parentLink:S-n-n-nn-<b>nnn</b></pre>
AgentVersion	<pre>keywords:agentVersion:va<b>lue</b></pre> <p>For example:</p> <pre>keywords:agentVersion:CentrifyDC 4.1</pre> <p><b>Note</b> This attribute is set as a <b>keywords</b> attribute if you are using Centrify, versions 3.0.x through 4.1.x. With the Centrify agent, version 4.2 or later, the <b>agentVersion</b> is stored using the <b>operatingSystemServicePack</b> attribute of the computer object.</p>
CpuCount	<pre>keywords:cpus:va<b>lue</b></pre> <p>For example:</p> <pre>keywords:cpus:2</pre> <p><b>Note</b> This attribute is only set if you are using Centrify, versions 3.0.x through 4.1.x. The attribute is not set or updated for computers with the Centrify agent, version 4.2 or later.</p>
JbossEnabled	<pre>keywords:jboss_enabled:va<b>lue</b></pre> <p>This attribute indicates whether the computer hosts JBoss applications.</p> <p>For example:</p> <pre>keywords:jboss_enabled:Fa<b>lse</b></pre>
TomcatEnabled	<pre>keywords:tomcat_enabled:va<b>lue</b></pre> <p>This attribute indicates whether the computer hosts Tomcat applications.</p> <p>For example:</p> <pre>keywords:tomcat_enabled:Fa<b>lse</b></pre>
UnixEnabled	<pre>keywords:unix_enabled:va<b>lue</b></pre> <p>For example:</p> <pre>keywords:unix_enabled:Tr<b>ue</b></pre>

Computer attribute	Stored in Active Directory attribute
webLogicEnabled	<code>keywords:weblogic_enabled:value</code>
	This attribute indicates whether the computer hosts WebLogic applications.
	For example: <code>keywords:jboss_enabled:True</code>
websphereEnabled	<code>keywords:websphere_enabled:value</code>
	This attribute indicates whether the computer hosts WebSphere applications.
	For example: <code>keywords:websphere_enabled:True</code>

## Classic RFC 2307 zones (3.x, 4.x)

The classic RFC 2307-compatible zone is similar to the classic Centrify zone, except that the data in the `serviceConnectionPoint` objects is associated with Active Directory user and group objects stored in RFC 2307-compliant attributes. For RFC 2307-compatible zones, Centrify makes use of a Windows Server feature, called Dynamic Auxiliary Classes, to dynamically bind `posixAccount` or `posixGroup` instances to the `serviceConnectionPoint` objects.



Binding the `posixAccount` or `posixGroup` to the user or group `serviceConnectionPoint` results in an Active Directory object with:

- Two object classes: the `serviceConnectionPoint` objectClass and the `posixAccount` or `posixGroup` objectClass.
- Two sets of attributes: those contributed by the `serviceConnectionPoint` object and those contributed by `posixAccount` or `posixGroup` object.



The structure of the zone and its sub-containers is the same as the classic Centrify zone layout, with each zone stored as a separate tree in the directory and sub-containers for the **Users**, **Groups**, and **Computers** in each zone, but you can use attributes from the `posixAccount` or `posixGroup` objectClass to store data in the RFC 2307-compliant format. Storing the data in RFC 2307-compliant attributes enables the information to be used by applications that conform to the RFC 2307 standard.

## Zone attributes in classic RFC 2307 zones

The zone object class and its attributes in the classic RFC 2307-compliant zone are the same as the classic Centrify zone. The zone object is stored as a container object, and the common name (cn) of the object must be set to the zone name. Most of the other attributes for a zone are stored as pseudo-attributes using the Active Directory description attribute.

The following table summarizes how zone attributes are stored in Active Directory for classic RFC 2307-compliant zones. For more information about any attribute setting, see [Zone attributes in classic Centrify zones](#).

Zone attribute	Stored in Active Directory attribute
ZoneName	cn:ZoneName For example: cn:default
ZoneVersion	displayName:ZoneVersion The valid values are: ZoneVersion <ul style="list-style-type: none"><li>■ <code>\$CimsZoneVersion3</code> for RFC 2307 zones, compatible with Centrify, version 3.x</li><li>■ <code>\$CimsZoneVersion4</code> for standard classic zones.</li><li>■ <code>\$CimsZoneVersion5</code> for classic RFC 2307 zones, compatible with Centrify, version 4.x</li></ul> For example: displayName:\$CimsZoneVersion5
Description	description:value For example: description:description:Pilot EMEA
NextUid	description:uidnext:value



Zone attribute	Stored in Active Directory attribute
	For example: description:uidnext:12098
NextGid	For example: description:gidnext:value
ReservedUids	For example: description:gidnext:12098 description:uidreserved:value
ReservedGids	For example: description:uidreserved:0-99:501 description:gidreserved:value
Availableshells	For example: description:gidreserved:1000-2500 description:availableshells:value
DefaultHomeDirectory	For example: description:availableshells:/bin/sh description:defaultthome:value
Defaultshell	For example: description:defaultthome:/nfs/\${user} description:defaultshell:value
DefaultGroup	For example: description:defaultshell:/bin/bash description:defaultgid:value
ZoneType	For example: description:defaultgid:12098 schema:Dynamic_Schema_Version The valid values for classic RFC 2307-compliant zones are: <ul style="list-style-type: none"><li>■ CDC_RFC_2307 (3.x compatible).</li><li>■ CDC_RFC_2307_2 (4.x compatible).</li></ul> For example: description: schema:CDC_RFC_2307

## User attributes in classic RFC 2307 zones

There are two object classes for the user extension object created in the users sub-container of the zone: the `serviceConnectionPoint` object class and the `posixAccount` object class.



User attribute	Stored in Active Directory attribute
	<code>cn:userlogin</code>
	and
UnixName	<code>uid:userlogin</code> For example: <code>uid:cain</code>
	<code>displayName:UserVersion</code>
UserVersion	This attribute determines compatibility between a user profile object and the Access Manager console. The only valid value for this attribute is <code>\$CimsUserVersion3</code> . For example: <code>displayName:\$CimsUserVersion3</code>
	<code>uidNumber:value</code>
Uid	For example: <code>uidNumber:458</code>
	<code>gidNumber:value</code>
Gid	For example: <code>gidNumber:458</code>
	<code>unixHomeDirectory:value</code>
Home	For example: <code>unixHomeDirectory:/home/shear</code>
	<code>loginShell:value</code>
Shell	For example: <code>loginShell:/bin/bash</code>
	<code>managedBy:DN_ActiveDirectoryUser</code>
	If the zone is a 2.x and 3.x compatible zone, you should set this attribute to the DN of the parent Active Directory user object. For example: <code>managedBy:cn=ben'tau,cn=users,dc=ice,dc=net</code>
ParentLink	If the zone does not need to be compatible with older versions of Centrify software, you can use the <code>keywords</code> attribute and <code>parentLink</code> pseudo-attribute to specify the security identifier (SID) of the parent Active Directory user object. For example: <code>keywords:parentLink:S-n-n-nn-nnn..</code>
UnixEnabled	<code>keywords:unix_enabled:value</code>



User attribute	Stored in Active Directory attribute
	For example: <code>keywords:unix_enabled:True</code>
	<code>keywords:foreign:value</code>
ForeignForest	This attribute indicates whether a user in a zone is from an external forest. For example: <code>keywords:foreign:False</code>

**Note:** The attribute name `unixHomeDirectory` is not RFC 2307-compliant. Microsoft used this name because the attribute `homeDirectory` was already used in Active Directory.

## Group attributes in classic RFC 2307 zones

There are two object classes for the group extension object created in the Groups sub-container of the zone: the `serviceConnectionPoint` object class and the `posixAccount` object class.

Group attribute	Stored in Active Directory attribute
	<code>cn:GroupName</code>
UnixName	For example: <code>cn:performx</code>
	<code>displayName:GroupVersion</code>
GroupVersion	This attribute determines compatibility between a group profile object and the Access manager console. The only valid value for this attribute is <code>\$CimsGroupVersion3</code> . For example: <code>displayName:\$CimsGroupVersion3</code>
	<code>gidNumber:value</code>
Gid	For example: <code>gidNumber:458</code>
	<code>managedBy:DN_ActiveDirectoryGroup</code>
ParentLink	If the zone is a 2.x and 3.x compatible zone, you should set this attribute to the DN of the parent Active Directory group object. For example: <code>managedBy:cn=interns,cn=users,dc=ice,dc=net</code> If the zone does not need to be compatible with older versions of Centrify



---

Group attribute	Stored in Active Directory attribute
	software, you can use the <code>keywords</code> attribute and <code>parentLink</code> pseudo-attribute to specify the security identifier (SID) of the parent Active Directory group object.  For example: <code>keywords:parentLink:S-n-n-nn-nnn..</code>
	<code>keywords:unix_enabled:value</code>
UnixEnabled	For example: <code>keywords:unix_enabled:True</code>
	<code>keywords:foreign:value</code>
ForeignForest	This attribute indicates whether a group in a zone is from an external forest.  For example: <code>keywords:foreign:False</code>

---

**Note:** The `posixGroup` group membership attributes are not set. Centrify uses the normal Active Directory mechanism for determining group membership.

## Computer attributes in classic RFC 2307 zones

A computer extension object is a `serviceConnectionPoint` object that is created in the `Computers` sub-container of the zone. The pseudo-attributes for this object are stored in the `keywords` attribute.

---

Computer attribute	Stored in Active Directory attribute
	<code>name:ComputerName</code>
UnixName	Normally, computer attribute values are set by the <code>adjoin</code> process and do not need to be modified.  For example: <code>name:magnolia.ajax.org</code>
	<code>displayName:ComputerVersion</code>
ComputerVersion	This attribute determines compatibility between a computer profile object and the Access Manager console. The only valid value for this attribute is <code>\$CimsComputerVersion3</code> .  For example: <code>displayName:\$CimsComputerVersion3</code>
ParentLink	<code>managedBy:DN_ActiveDirectoryGroup</code>

---



Computer attribute	Stored in Active Directory attribute
	<p>If the zone is a 2.x and 3.x compatible zone, you should set this attribute to the DN of the parent Active Directory computer object.</p> <p>For example:</p> <pre>managedBy:cn=hr,cn=computers,dc=ajax,dc=org</pre> <p>If the zone does not need to be compatible with older versions of Centrify software, you can use the <b>keywords</b> attribute and <b>parentLink</b> pseudo-attribute to specify the security identifier (SID) of the parent Active Directory computer object.</p> <p>For example:</p> <pre>keywords:parentLink:S-n-n-nn-nnn..</pre>
AgentVersion	<pre>keywords:agentVersion:value</pre> <p>For example:</p> <pre>keywords:agentVersion:CentrifyDC 4.1</pre> <p><b>Note</b> This attribute is set as a <b>keywords</b> attribute if you are using Centrify, versions 3.0.x through 4.1.x. With the Centrify agent, version 4.2 or later, the <b>AgentVersion</b> is stored using the <b>operatingSystemServicePack</b> attribute of the computer object.</p>
CpuCount	<pre>keywords;cpus:value</pre> <p>For example:</p> <pre>keywords:cpus:2</pre> <p><b>Note</b> This attribute is only set if you are using Centrify, versions 3.0.x through 4.1.x. The attribute is not set or updated for computers with the Centrify agent, version 4.2 or later.</p>
JbossEnabled	<pre>keywords:jboss_enabled:value</pre> <p>This attribute indicates whether the computer hosts JBoss applications.</p> <p>For example:</p> <pre>keywords:jboss_enabled:False</pre>
TomcatEnabled	<pre>keywords:tomcat_enabled:value</pre> <p>This attribute indicates whether the computer hosts Tomcat applications.</p> <p>For example:</p> <pre>keywords:tomcat_enabled:False</pre>
UnixEnabled	<pre>keywords:unix_enabled:value</pre> <p>For example:</p> <pre>keywords:unix_enabled:True</pre>
webLogicEnabled	<pre>keywords:weblogic_enabled:value</pre>



Computer attribute	Stored in Active Directory attribute
	This attribute indicates whether the computer hosts WebLogic applications. For example: <code>keywords:WEBLOGIC_enabled:True</code>
<code>websphereEnabled</code>	This attribute indicates whether the computer hosts WebSphere applications. For example: <code>keywords:websphere_enabled:True</code>

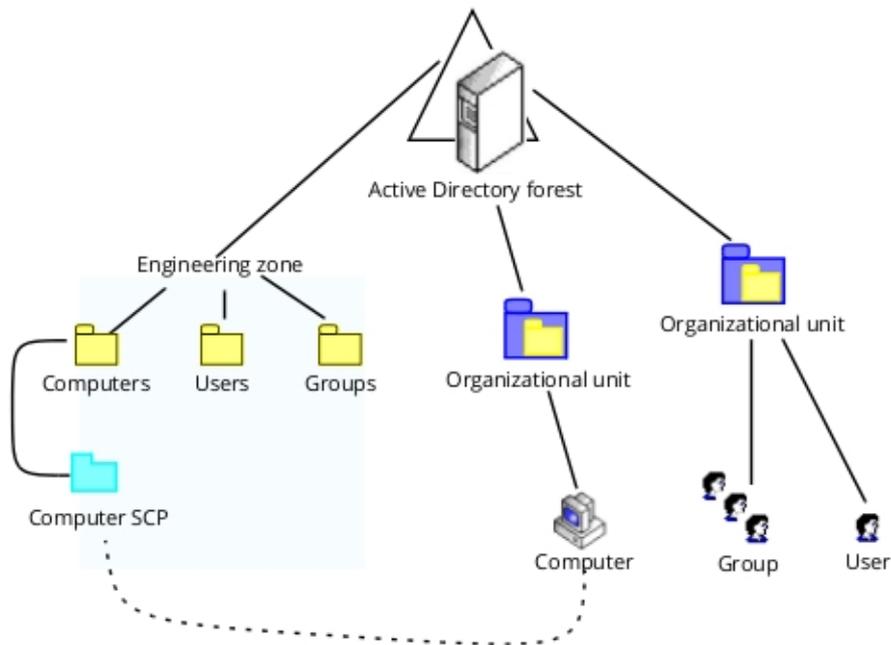
## Classic SFU-compliant zones (version 3.5)

If you have the Microsoft Services for UNIX (SFU) schema extension installed, you have the option of using SFU-compliant zones for storing data. With SFU-compliant zones, UNIX-specific attributes for users and groups are stored in the actual Active Directory user and Active Directory group objects, using attributes in Microsoft Services For UNIX (SFU) schema extension.

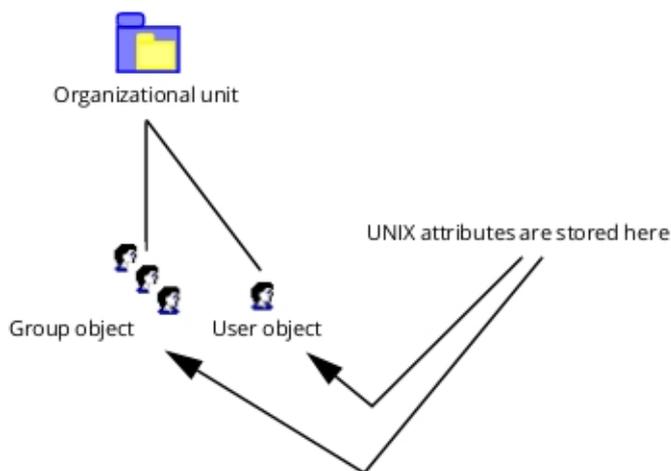
**Note:** The schema extension must be already be installed in the forest. You cannot create SFU-compliant zones if the schema extension is not installed.

Unlike standard Centrify zones, where a single Active Directory user can have multiple UNIX profiles, a single Active Directory user can only exist in one SFU zone because there is only one set of attributes in the Active Directory user object. A single user can, however, be in any number of Centrify zones and zero or one SFU zone.

The structure of the zone and its sub-containers is the same as the classic Centrify zone layout, with each zone stored as a separate tree in the directory and sub-containers for the **Users**, **Groups**, and **Computers** in each zone, but only the Computers sub-container is used.



Unlike classic Centrify zones, in which UNIX attributes are stored in the serviceConnectionPoint objects, the SFU zones store UNIX attributes in the User and Group objects and use attributes provided by the SFU schema extension.



## Zone attributes in classic SFU-compliant zones

The zone object class and its attributes in the classic SFU zone are similar to the classic Centrify zone, except that the zone must also include the NIS domain name and domain attributes. Like the classic Centrify zones, the zone object is stored as a container object, and the common name (cn) of the object must be



set to the zone name. Most of the other attributes for a zone are stored as pseudo-attributes using the Active Directory `description` attribute.

The following table summarizes how zone attributes are stored in Active Directory for SFU-compliant zones. For more information about any attribute setting, see [Zone attributes in classic Centrify zones](#).

Zone attribute	Stored in Active Directory attribute
ZoneName	<code>cn:ZoneName</code> For example: <code>cn:default</code>
ZoneVersion	<code>displayName:ZoneVersion</code> The only valid value is <code>\$CimsZoneVersion2</code> . For example: <code>displayName:\$CimsZoneVersion2</code>
Description	<code>description:description:value</code> For example: <code>description:description:Pilot EMEA</code>
NextUid	<code>description:uidnext:value</code> For example: <code>description:uidnext:12098</code>
NextGid	<code>description:gidnext:value</code> For example: <code>description:gidnext:12098</code>
ReservedUids	<code>description:uidreserved:value</code> For example: <code>description:uidreserved:0-99:501</code>
ReservedGids	<code>description:gidreserved:value</code> For example: <code>description:gidreserved:1000-2500</code>
Availableshells	<code>description:availableshells:value</code> For example: <code>description:availableshells:/bin/sh</code>
DefaultHomeDirectory	<code>description:defaulthome:value</code> For example: <code>description:defaulthome:/nfs/\${user}</code>
DefaultShell	<code>description:defaultshell:value</code> For example:



Zone attribute	Stored in Active Directory attribute
	<code>description:defaultshell:/bin/bash</code>
	<code>description:defaultgid:value</code>
DefaultGroup	For example: <code>description:defaultgid:12098</code>
	<code>description:schema:Dynamic_Schema_Version</code>
ZoneType	The only valid value for SFU zones is: <ul style="list-style-type: none"><li>■ <code>SFU_3_0</code> for the Microsoft Services for UNIX (SFU) versions 3.x or 4.x schema extension.</li></ul> For example: <code>description:schema:SFU_3_0</code>
	<code>description:Nisdomain:value</code>
NisDomain	This attribute describes the NIS domain for that defines the scope of the zone. For more information about this setting, see the User object attributes. For example: <code>description: Nisdomain:XXX</code>
	<code>description:Sfudomain:value</code>
SFUDomain	The SFU domain contains the users and groups. The members of an SFU domain can only come from one domain. For example: <code>description:Sfudomain:mfg.ajax.org</code>

## User attributes in classic SFU-compliant zones

In classic SFU zones, UNIX-specific user attributes are stored as part of the Active Directory user object.

User attribute	Stored in Active Directory attribute
	<code>MSSFU30Name:userlogin</code>
UnixName	For example: <code>MSSFU30Name:cain</code>
	<code>MSSFU30UidNumber:value</code>
Uid	For example: <code>MSSFU30UidNumber:458</code>
	<code>MSSFU30GidNumber:value</code>
Gid	For example:



User attribute	Stored in Active Directory attribute
	MSSFU30GidNumber:458
Home	MSSFU30HomeDirectory:value For example: MSSFU30HomeDirectory:/home/shear
Shell	MSSFU30Shell:value For example: MSSFU30Shell:/bin/bash
NisDomain	MSSFU30NisDomain:value  This attribute must be defined. Centrify uses this setting to determine if the user is a member of the zone. When you create SFU-compliant zones, you must specify the NIS domain name that should be included. For example, you can configure <b>zone_beijing</b> to include all users and groups whose NIS domain attribute is set to <b>nisbeijing</b> .  For example: MSSFU30NisDomain:nisbeijing.local
UnixEnabled	Not supported.

## Group attributes in classic SFU-compliant zones

In classic SFU-compliant zones, UNIX-specific group attributes are stored as part of the Active Directory group object.

Group attribute	Stored in Active Directory attribute
UnixName	MSSFU30Name:GroupName For example: MSSFU30Name:performx
Gid	MSSFU30GidNumber:value For example: MSSFU30GidNumber:458
NisDomain	MSSFU30NisDomain:value  This attribute must be defined. Centrify uses this setting to determine if the group is a member of the zone. When you create SFU-compliant zones, you must specify the NIS domain name that should be included. For example, you can configure <b>zone_beijing</b> to include all users and groups whose NIS domain attribute is set to <b>nisbeijing</b> .  For example:



Group attribute	Stored in Active Directory attribute
	MSSFU30nisDomain:nisbejing.local
UnixEnabled	Not supported.

**Note:** The Microsoft Services for UNIX schema extension supports group membership as an attribute of the group object in the same way the RFC 2307-compliant schema does. Centrify does not use this attribute, however. Centrify uses Active Directory group membership to identify group members.

## Classic SFU-compliant zones (version 4.0)

if you have the Microsoft Services for UNIX (SFU), version 4.0, schema extension installed, you have the option of using SFU-compliant zones for storing data. Centrify SFU-compliant zones for the Microsoft Services for UNIX (SFU), version 4.0, schema extension are similar to SFU-compliant zones for version 3.5, except that Microsoft Services for UNIX (SFU), version 4.0, uses the R2 schema. The UNIX-specific attributes for users and groups are still stored in the actual Active Directory user and Active Directory group objects, but use the R2 schema attributes instead of the mssfU\* attributes used in Microsoft Services For UNIX (SFU), version 3.5.

**Note:** You can only create this type of zone if the R2 schema is installed. If the R2 schema attributes are not available, you cannot create this type of zone.

## Zone attributes in classic SFU 4.0 zones

The zone object class and its attributes in the classic SFU-compliant zones for the Microsoft Services for UNIX (SFU), version 4.0, schema extension are the same as described in [Zone attributes in classic SFU-compliant zones](#). For more information about any attribute setting, see [Zone attributes in classic Centrify zones](#).



## User attributes in classic SFU 4.0 zones

In classic SFU-compliant zones, UNIX-specific attributes are stored as part of the Active Directory user object. With Microsoft Services for UNIX, version 4.0, however, the R2 schema attributes are used.

User attribute	Stored in Active Directory attribute
	<code>uid:userlogin</code>
UnixName	For example: <code>uid:cain</code>
	<code>uidNumber:value</code>
Uid	For example: <code>uidNumber:458</code>
	<code>gidNumber:value</code>
Gid	For example: <code>gidNumber:458</code>
	<code>unixHomeDirectory:value</code>
Home	For example: <code>unixHomeDirectory:/home/shear</code>
	<code>loginShell:value</code>
Shell	For example: <code>loginShell:/bin/bash</code>
	<code>MSSFU30nisDomain:value</code>
NisDomain	This attribute must be defined. Centrify uses this setting to determine if the user is a member of the zone. When you create SFU-compliant zones, you must specify the NIS domain name that should be included. For example, you can configure <code>zone_beijing</code> to include all users and groups whose NIS domain attribute is set to <code>nisbeijing</code> .  For example: <code>MSSFU30nisDomain:nisbeijing.local</code>
UnixEnabled	Not supported.

## Group attributes in classic SFU 4.0 zones

In classic SFU-compliant zones, UNIX-specific attributes are stored as part of the Active Directory group object. With Microsoft Services for UNIX, version 4.0, however, the R2 schema attributes are used.

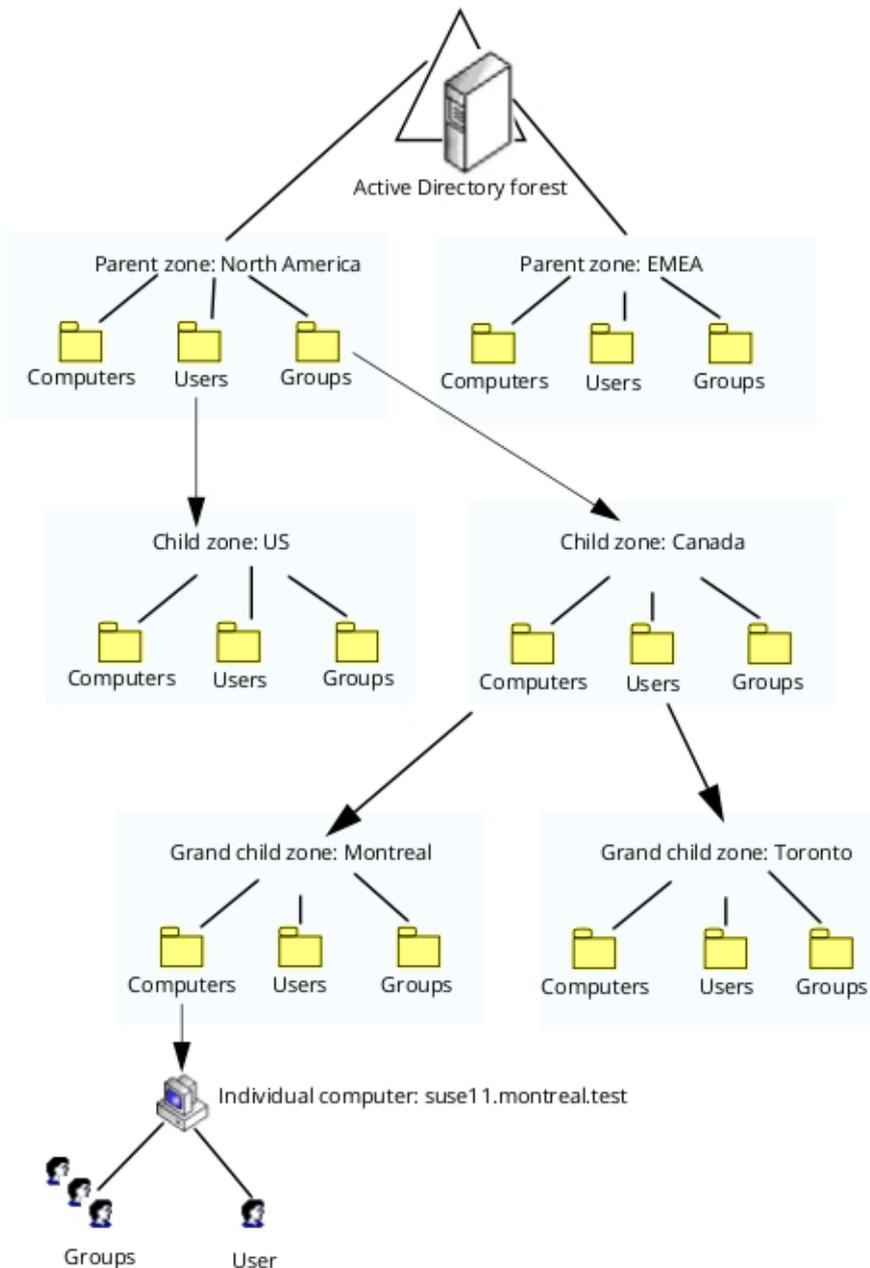


Group attribute	Stored in Active Directory attribute
	<code>cn:GroupName</code>
UnixName	For example: <code>cn:performx</code>
Gid	For example: <code>gidNumber:458</code>
NisDomain	<code>MSSFU30NisDomain:value</code>  This attribute must be defined. Centrify uses this setting to determine if the group is a member of the zone.  When you create SFU-compliant zones, you must specify the NIS domain name that should be included. For example, you can configure <code>zone_beijing</code> to include all users and groups whose NIS domain attribute is set to <code>nisbeijing</code> .  For example: <code>MSSFU30NisDomain:nisbeijing.local</code>
UnixEnabled	Not supported.

## Hierarchical Centrify zones (5.x)

In hierarchical Centrify zones, each zone is part of a tree of zones and Active Directory containers for users, groups, and computers. The effective profile for each user, group, or computer in a hierarchical zone is determined by attributes defined in the current zone, in parent zones, and in zone default values, with values lower in the hierarchy taking priority. In addition, individual computers can have computer-level overrides of users, groups, and role assignments. When you assign computer-level overrides for a specific computer, internally Centrify creates a *computer-specific zone* that contains the users, groups, and computer role assignments that are specific to only that one computer. Computer zones are not exposed as zones in Access Manager, but are treated as end nodes in the zone hierarchy.

The following figure illustrates the basic structure used for hierarchical Centrify zones. Attributes are inherited from higher-level to lower-level zones as indicated by the arrows in the figure.



In hierarchical zones, because User and Zone objects inherit from their parent zones, it is not necessary for every hierarchical level to have a full set of attributes. If a user profile is incomplete after inheriting all the ancestor's attributes, missing mandatory attributes are filled from defaults. Missing UIDs are generated from an RID-based algorithm.

See [Classic Centrifly zones \(2.x, 3.x, 4.x\)](#) for a general description of the way Centrifly attributes are stored in Active Directory.



## Zone attributes in standard hierarchical zones

The zone object class is stored as a container object. The common name (cn) of the object must be set to the zone name. Most of the other attributes for a zone are stored as pseudo-attributes using the Active Directory `description` attribute. The following table summarizes how zone attributes are stored in Active Directory for hierarchical Centrify zones.

Zone attribute	Stored in Active Directory attribute	Inherited
ZoneName	<code>cn:ZoneName</code> For example: <code>cn:global</code>	No
Description	<code>description:description:value</code> For example: <code>description:description:Pilot-NA</code>	No
AvailableShells	<code>description:availableshells:shell1:shell2</code> For example: <code>description:availableshells:/bin/sh</code>	Yes
DefaultShell	<code>description:defaultshell:value</code> or <code>description:defaultshell:%{shell}</code> For example: <code>description:defaultshell:/bin/bash</code>	Yes
DefaultHomeDirectory	<code>description:defaulthome:value</code> or <code>description:defaulthome:%{home}/%{user}</code> For example: <code>description:defaulthome:/nfs/jsmith</code>	Yes
UserDefaultGecos	<code>description:defaultgecos:\${u:cn}</code> For example: <code>description:defaulttgecos:\${u:upn}</code>	Yes
customVariable	<code>description:%variablename:value</code> One for each variable. For example: <code>description:%admin:SAMAccountName</code>	Yes
ReservedUids	<code>description:uidreserved:value</code> This attribute can be a multi-valued list, using a colon as the separator. Values can be individual numbers or	Yes



Zone attribute	Stored in Active Directory attribute	Inherited
	<p>a range of numbers separated with a dash character (-).</p> <p>For example:</p> <p>description:uidreserved:0-99:501</p>	
ReservedGids	<p>This attribute has the same format as the <b>reserveduids</b> attribute. For example:</p> <p>description:gidreserved:1000-2500</p>	Yes
UserDefaultuid	<p>description:defaultuid:value</p> <p>Set value to <code>\${uidnext}</code> to use the zone's cram attribute <b>uidnext</b>. The cram attribute is where the key-value pairs ("name:value") are stored.</p> <p>Set value to <code>\${autosid}</code> to generate the UID from the domain SID and user RID.</p> <p>For example:</p> <p>description:defaultuid:\${autosid}</p>	Yes
DefaultGroup	<p>description:defaultgid:value</p> <p>Set value to <code>-1</code> to use private groups.</p> <p>For example:</p> <p>description:defaultgid:12098</p>	Yes
UserDefaultName	description:username:\${u:SAMAccountName}	Yes
UserDefaultRole	description:defaultrole:role-name	Yes
GroupDefaultGid	<p>description:defaultgroupgid:value</p> <p>Set value to <code>\${gidnext}</code> to use the zone's cram attribute <b>gidnext</b> in classic zones.</p> <p>Set value to <code>\${autosid}</code> to generate the GID from the domain SID and group RID in hierarchical zones.</p> <p>For example:</p> <p>description:defaultgid:\${autosid}</p>	Yes
GroupDefaultName	description:groupname:\${g:CN}	Yes
NISDomain	description:nisdomain:name	Yes
Schema	<p>description:schema:name</p> <p>Possible values are:</p> <ul style="list-style-type: none"> <li>■ CDC_RFC_2307 (for a classic RFC 2307 zone)</li> <li>■ CDC_GENERIC (for a classic Centrifly zone)</li> <li>■ SFU_3_0 (For a classic SFU-compliant R2)</li> </ul>	No



Zone attribute	Stored in Active Directory attribute	Inherited
	schema zone) <ul style="list-style-type: none"> <li>■ SFU_3_0V1 (For a classic SFU-compliant zone)</li> </ul> For example: description:Cchema:DC_GENERIC	
AgentlessAttribute	For example: description:pwsync:msSFU30Password	Yes
Licenses	description:license:guid description:alternateDomain:domain.name	Yes
SFUDomain	This is a multi-value attribute. Multi-value attributes are possible because the keyword and value are combined, making each line of the description-keyword string unique. description:parentLink:MS-GUID@DOMAIN.NAME	Yes
Parent	For example: samAccountName@domain.name[:N]: "joe@ajax.com"	No
objectType	displayName=\$CimsZoneVersionnumber where the zone version number can be: <ul style="list-style-type: none"> <li>■ \$CimsUserVersion4 for a Centrify zone</li> <li>■ \$CimsUserVersion5 for a RFC 2307 zone</li> </ul>	No

## User attributes in hierarchical zones

A user extension object is a serviceConnectionPoint object that is created in the Users sub-container of the zone. The pseudo-attributes for this object are stored in the keywords attribute.

User attribute	Stored in Active Directory attribute	Inherited
cn	sAMAccountName@domain.name[:N]	No
objectType	displayName=\$CimsUserVersion4 keywords:login:name	No
Name	For example: keywords:login:cain	Yes
uid	keywords:uid:value For example:	Yes



User attribute	Stored in Active Directory attribute	Inherited
	keywords:uid:458	
	keywords:gid:va_lue	
Gid	For example: keywords:gid:458	Yes
	keywords:home:va_lue	
Home	For example: keywords:home:/home/she_a	Yes
	keywords:shell:va_lue	
shell	For example: keywords:shell:/bin/bash	Yes
	gecos:va_lue	
Gecos	For example: gecos:%{u:displayName}	Yes

User and group extended attributes are specific to a particular computer and can be set on a per-user or per-group basis. The format for extended attributes depend on the format required for a particular operating system. Currently, only AIX extended attributes are supported.

Each attribute name starts with a prefix that indicates the operating system to which it applies (for example, aix.) and is followed by the attribute name. The valid values for each attribute depend on the attribute type, and can be a string, number or Boolean value. Attributes that support multiple values are specified with separate name-value pairs.

The specific user and group extended attributes that are available for you to set depend on the version of the operating system running on the computer where the attributes are used. For detailed information about the extended attributes available and valid values on a specific version of the AIX operating system, see your AIX documentation.

The following table lists some of the most commonly-used **user extended attributes** for illustration purposes. It does not represent the complete list of user and group extended attributes that might be available on any given version of the operating system.

Extended attribute	Description
aix.admin	Specifies the administrative status of the user as <code>true</code> or <code>false</code> .
aix.admgroups	Lists the groups that the user administrates as a comma-separated list of



Extended attribute	Description
	group names.
<code>aix.daemon</code>	Specifies whether the user can execute programs using the the <code>cron</code> daemon or the system resource controller ( <code>src</code> ).
<code>aix.rlogin</code>	Specifies whether the user account can be logged into remotely using <code>telnet</code> or <code>rlogin</code> .
<code>aix.su</code>	Indicates whether other users can switch to the user account with the <code>su</code> command.
<code>aix.sugroups</code>	Lists the groups can switch to the user account as a comma-separated list of group names.
<code>aix.tpath</code>	Indicates the user's trusted path status.
<code>aix.ttys</code>	Lists the terminals that can access the account as a comma-separated list of full path names, or using <code>ALL</code> to indicate all terminals.
<code>aix.fsize</code>	Sets the soft limit for the largest file a user's process can create or extend or a value of <code>-1</code> to specify <code>unlimited</code> for this attribute.
<code>aix.core</code>	Sets the soft limit for the largest core file a user's process can create or a value of <code>-1</code> to specify <code>unlimited</code> for this attribute.
<code>aix.cpu</code>	Sets the soft limit for the maximum number of seconds of system time that a user's process can use or a value of <code>-1</code> to specify <code>unlimited</code> for this attribute.
<code>aix.data</code>	Sets the soft limit for the size of a user's data segment or a value of <code>-1</code> to specify <code>unlimited</code> for this attribute
<code>aix.rss</code>	Sets the soft limit for the largest amount of physical memory a user's process can allocate or a value of <code>-1</code> to specify <code>unlimited</code> for this attribute.
<code>aix.stack</code>	Sets the soft limit for the largest process stack segment for a user's process or a value of <code>-1</code> to specify <code>unlimited</code> for this attribute.
<code>aix.nofiles</code>	Sets the soft limit for the number of file descriptors a user process can have open at one time or a value of <code>-1</code> to specify <code>unlimited</code> for this attribute.
<code>aix.umask</code>	Determines file permissions for the user using a three-digit octal value such as <code>022</code> .

## Group attributes in hierarchical zones

A group extension object is a `serviceConnectionPoint` object that is created in the `Groups` sub-container of the zone. The pseudo-attributes for this object are stored in the `keywords` attribute.



Group attribute	Stored in Active Directory attribute	Inherited
version	displayName=\$CimsZoneVersion4 or, for RFC 2307 objects, use version 5: displayName=\$CimsZoneVersion5	No
name	keywords:login:Name For example: keywords:login:ibmdba	Yes
gid	keywords:gid:value For example: keywords:gid:458 If the group is in a standard zone, the GID is stored as <b>gid:xxx</b> in the <b>keywords</b> attribute. If the group is in an RFC 2307 zone, the GID is stored in the schema's <b>gid</b> attribute.	Yes
parentLink	keywords:parentLink:MS-SID For example: keywords:parentLink:S-1-5-21-387451290	No
IsMembershipRequired	keywords:required:value For example: keywords:required:true	Yes
InheritFromParent	keywords:inherit:value For example: keywords:inherit:true	No

## Computer attributes in hierarchical zones

A computer extension object is a `serviceConnectionPoint` object that is created in the `Computers` sub-container of the zone. The pseudo-attributes for this object are stored in the `keywords` attribute.

Computer attribute	Stored in Active Directory attribute
unixName	cn:name:ComputerName Normally, the computer attribute values are set by the <code>adjoin</code> process and do not need to be modified.
schemaVersion	displayName:ComputerVersion This attribute determines compatibility between a computer profile object



Computer attribute	Stored in Active Directory attribute
	and the Access Manager console. The only valid value for this attribute is <code>\$CimsComputerVersion3</code> .
	For example: <code>displayName:\$CimsComputerVersion3</code>
	<code>keywords:agentVersion:Version</code>
agentVersion	For example: <code>keywords:agentVersion:CentrifyDC 5.4.0-197</code>
	<code>keywords:parentLink:MS-SID</code>
parentLink	For example: <code>keywords:parentLink:S-1-5-21-387451290-2189</code>

## Computer-specific zone attributes in standard hierarchical zones

A computer zone object is a zone object that contains computer-specific users and groups. This object is a special type of hierarchical Zone container with `computerName:zone` and `version displayName=$CimsComputerZoneVersion1`.

## User attributes in RFC 2307-compliant zones

If you create a hierarchical zone when using the RFC 2307-compliant schema, user attributes are stored in much the same way as in a classic zone with user object attributes stored in corresponding Active Directory attributes.

User attribute	Stored in Active Directory attribute	Inherited
cn	<code>sAMAccountName@domain.name[:N]</code>	No
Version	<code>displayName=\$CimsUserVersion5</code>	No
Name	<code>uid</code>	Yes
Uid	<code>uidNumber</code>	Yes
Gid	<code>gidNumber</code>	Yes
Home	<code>unixHomeDirectory</code>	Yes
Shell	<code>loginShell</code>	Yes
Gecos	<code>gecos</code>	Yes

For more information, see [User attributes in classic RFC 2307 zones](#).



## Group attributes in RFC 2307-compliant zones

If you create a hierarchical zone when using the RFC 2307-compliant schema, group attributes are stored in much the same way as in a classic zone with group object attributes stored in corresponding Active Directory attributes.

User attribute	Stored in Active Directory attribute	Inherited
Version	displayName=\$CimsUserVersion5	No
Name	keywords=login:Name	Yes
Gid	gidNumber	Yes
Gecos	gecos	Yes

For more information, see [Group attributes in classic RFC 2307 zones](#).

## User attributes in hierarchical SFU zones

If you create a hierarchical zone when using the Microsoft Services for UNIX (SFU) schema, user attributes are stored in the same way as in a classic SFU zone with the following exceptions:

- If the SFU schema is version 3.x, the Gecos field is stored in the mssfu30gecos Active Directory attribute.
- If the SFU schema is version 4.x, the Gecos field is stored in the gecost Active Directory attribute.

For more information, see [User attributes in classic SFU-compliant zones](#).

## Group attributes in hierarchical SFU zones

If you create a hierarchical zone when using the Microsoft Services for UNIX (SFU) schema, group attributes are stored in the same way as in a classic SFU zone. For more information, see [Group attributes in classic SFU-compliant zones](#).

## Using commands and scripts to perform tasks

With an understanding of how the data is stored in Active Directory for different zones types, you can use ADEdit or LDAP commands to perform a



wide range of tasks from the UNIX command line or in scripts and custom programs. The following examples illustrate how you can use the OpenLDAP command line interface (CLI) that is installed with the Centrify agent to perform administrative tasks. The OpenLDAP CLI is Kerberos-enabled, so it is not necessary to supply credentials. All operations run with the permissions of the Active Directory user currently logged in. For information about using ADEdit to perform administrative tasks, see the *ADEdit Command Reference and Scripting Guide*.

## Getting started

The OpenLDAP commands provided with the Centrify agent package support all of the standard command line options, plus some additional options to make it easier to use them to work with Active Directory. For example, the LDAP commands packaged with the Centrify agent accept a URL of the form:

```
LDAP://domain_name
```

to specify the nearest domain controller in the specified domain, or a URL of:

```
LDAP://
```

In addition to the LDAP commands, Centrify includes several other command line programs and environment variables you may find useful in creating scripts to perform administrative tasks. For example, your scripts can take advantage of the environment variables that are set for an Active Directory user upon authentication. You can also use commands such as `adinfo` and `adfinddomain` to return information or supply input for administrative scripts.

On Linux and UNIX computers with a Centrify agent, version 5.0 or later, you can use the ADEdit command line utility and library of commands to perform administrative tasks instead of using LDAP commands or single-purpose commands. For information about using ADEdit, see the *ADEdit Command Reference and Scripting Guide*.

## Creating a classic zone

The following example shows the commands and data needed to create a classic Centrify zone named “zone1”. Zone creation is almost identical for all zone types. Only the value of `displayName` and the schema pseudo-attribute differ from zone type to zone type.



Before you can create the zone itself, however, you must create an Active Directory container with the appropriate properties. The zone container must also contain four other sub-containers to accommodate the UNIX attributes for Computers, Users, Groups, and NISMaps for the zone. You can create your zone anywhere within the directory tree.

To create a zone container and zone properties using the `ldapadd` command:

```
ldapadd -H ldap://mydc.acme.com << END_DATA
```

```
# Add the zone container
dn: cn=zone1,cn=myzones,dc=acme,dc=com
objectClass: container
cn: zone1
description: uidnext:10005
description: gidnext:10007
description: gidreserved:0-99
description: uidreserved:0-99
description: availableshells:/bin/bash:/bin/csh:/bin/sh:/bin/tcsh
description: defaulthome:/home/${user}
description: privategroupcreation:True
description: defaultshell:/bin/bash
description: schema:Dynamic_Schema_3_0
displayName: \${CimsZoneVersion2}
showInAdvancedViewOnly: TRUE
name: default

# Add the Computers sub-container
dn: CN=Computers, cn=zone1,cn=myzones,dc=acme,dc=com
objectClass: container
cn: Computers
showInAdvancedViewOnly: TRUE
name: Computers

# Add the Groups sub-container
dn: CN=Groups, cn=zone1,cn=myzones,dc=acme,dc=com
objectClass: container
cn: Groups
showInAdvancedViewOnly: TRUE
name: Groups

# Add the Users sub-container
dn: CN=Users, cn=zone1,cn=myzones,dc=acme,dc=com
objectClass: container
cn: Users
showInAdvancedViewOnly: TRUE
name: Users

# Add the NISMaps sub-container
dn: CN=NisMaps, cn=zone1,cn=myzones,dc=acme,dc=com
objectClass: container
cn: NisMaps
showInAdvancedViewOnly: TRUE
name: NisMaps
END_DATA
```



## Add a user to a classic zone

Adding a UNIX user or group profile to an Active Directory user or group object requires you to know the security identifier (SID) for the Active Directory user or Active Directory group. This information is necessary to link the UNIX attributes in the UNIX profile to its corresponding Active Directory account. One way to get this information is to use the Windows Server directory service command-line tool `dsquery` to return the SID for a specific user:

```
dsquery user -samid user | dsget user -sid -samid
```

For example, to list the `samAccountName` and SID for the user with the `samAccountName` `jane`:

```
dsquery user -samid jane | dsget user -sid -samid
```

**Note:** For more information on using `dsquery`, search for the command on the Microsoft website.

Once you have identified the SID for a user or group, you can use the `ldapadd` command to add a profile for the user or group to the zone.

The following example illustrates how to add user “joe” to “zone1” where “zone1” is a classic RFC 2307-compliant zone:

```
ldapadd -H ldap://mydc.acme.com << END_DATA
dn: CN=joe,CN=Users,cn=zone1,cn=myzones,dc=acme,dc=com
objectClass: posixAccount
objectClass: serviceConnectionPoint
cn: joe
displayName: \${CimsUserVersion}3
showInAdvancedViewOnly: TRUE
name: joe
keywords: unix_enabled:True
keywords: parentLink:S-1-5-21-397955417-626881126-188441444-512
uid: joe
uidNumber: 123
gidNumber: 234
unixHomeDirectory: /home/joe
loginShell: /bin/bash
END_DATA
```

The following example illustrates how to add the user profile “joe” to “zone1” where “zone1” is a Standard zone:

```
ldapadd -H ldap://mydc.acme.com << END_DATA
dn: CN=joe,CN=Users,cn=zone1,cn=myzones,dc=acme,dc=com
objectClass: serviceConnectionPoint
cn: joe
displayName: \${CimsUserVersion}2
showInAdvancedViewOnly: TRUE
name: joe
```



```
keywords: unix_enabled:True
keywords: parentLink:S-1-5-21-397955417-626881126-188441444-512
keywords: uid:123
keywords: gid:234
keywords: home:/home/joe
keywords: shell:/bin/bash
END_DATA
```



# Adding users in a one-way trust environment

This chapter explains how to add a user in a one-way trust environment by using the Centrify Windows API.

To add a user in a one-way trust environment, follow these steps:

1. Select an account in a domain that is in a one-way trust relationship with the remote forest so that the account has access to resources in both domains.

For example, suppose the corporate domain `company.corp.com` is trusted by the remote domain `companyDMZ.com`, which is where you intend to add a user. Select an account in the `company.corp.com` domain that can access resources in the `companyDMZ.com` domain.

2. Verify that the selected account has permission to modify a zone.

You can use the zone delegation wizard to add this permission to the selected account. By default, if the user account is a member of the Domain Administrators group in `companyDMZ.com`, you have the necessary permissions.

3. Use `Cims.Connect()` to connect to the `companyDMZ.com` domain to get the `Cims` object.
4. Obtain an `IADsUser` object for the remote forest user that you will add to the zone.

To obtain an `IADsUser` for `company.corp.com` using VBScript, for example, use the following code:

```
u = GetObject  
(LDAPCOMPANY.CORP.NETCN=UserName,CN=Users,DC=wonder,DC=land)
```

If you log in as a domain user from `company.corp.com`, you should have sufficient permission.



5. Get the user object by passing the `IADsUser` object you obtained in the previous step to `cmds.GetUser(x)`.
6. With the user object, you can use `User.AddUnixProfile()` to add the zone profile.



# Reading and setting timebox values

A role specifies a collection of rights. A role object contains a field, `timebox`, that defines the hours and days of the week that a role is either enabled or disabled. Setting the `timebox` field in a role object defines when a role's rights are in effect.

You can read and set a role's `timebox` field using the `Role.ApplicableTimeHexString` property. You can modify an existing `timebox` value one day or one hour at a time using the `Role.GetSshRights` and `Role.SetApplicableHour` methods.

To interpret a `timebox` value, or to set it directly, however, you must know the `timebox` value format. This appendix explains the format.

## Hex string

The `timebox` value is a 42-character (21-byte) hexadecimal value stored as a string. When the hex value is converted to a binary value, its 168 bits each map to a single hour within the week. If a bit is set to 1, its corresponding hour is enabled for the role. If set to 0, its corresponding hour is disabled.

## Hour mapping

Each day of the week takes three bytes (24 bits) to specify how its hours are enabled or disabled. The following tables show how the hours of a day are mapped to the bits within each of a day's three bytes.

For byte 0 of each day of the week, you can enable or disable the hours a role is available from midnight to 8:00 AM:



Hour	Bit
12-1 AM	0 (least-significant bit)
1-2 AM	1
2-3 AM	2
3-4 AM	3
4-5 AM	4
5-6 AM	5
6-7 AM	6
7-8 AM	7 (most-significant bit)

For byte 1 of each day of the week, you can enable or disable the hours a role is available from 8:00 AM to 4:00 PM:

Hour	Bit
8-9 AM	0 (least-significant bit)
9-10 AM	1
10-11 AM	2
11-12 AM	3
12-1 PM	4
1-2 PM	5
2-3 PM	6
3-4 PM	7 (most-significant bit)

For byte 2 of each day of the week, you can enable or disable the hours a role is available from 4:00 PM to midnight:

Hour	Bit
4-5 PM	0 (least-significant bit)
5-6 PM	1
6-7 PM	2
7-8 PM	3
8-9 PM	4
9-10 PM	5
10-11 PM	6
11-12 PM	7 (most-significant bit)



## Day mapping

Each of the seven days in a week have three bytes within the 21-byte timebox value. These bytes are in chronological order from most-significant byte to least-significant byte. (Note that this is the opposite of chronological bit order within each byte, which is LSB to MSB.)

The starting point of a week is 4 PM on Saturday afternoon.

The table below shows how each day's three bytes (0-2) map to the timebox value's bytes, listed here in order from most-significant byte to least-significant byte.

Day byte	Timebox value byte
Saturday, byte 2	20 (most-significant byte)
Sunday, byte 0	19
Sunday, byte 1	18
Sunday, byte 2	17
Monday, byte 0	16
Monday, byte 1	15
Monday, byte 2	14
Tuesday, byte 0	13
Tuesday, byte 1	12
Tuesday, byte 2	11
Wednesday, byte 0	10
Wednesday, byte 1	9
Wednesday, byte 2	8
Thursday, byte 0	7
Thursday, byte 1	6
Thursday, byte 2	5
Friday, byte 0	4
Friday, byte 1	3
Friday, byte 2	2
Saturday, byte 0	1
Saturday, byte 1	0 (least-significant byte)