



Server Suite

Windows API Programmer Guide

Version: 2024.x

Publication Date: 7/14/2025

Server Suite Windows API Programmer Guide

Version: 2024.x, Publication Date: 7/14/2025

© Delinea, 2025

Warranty Disclaimer

DELINEA AND ITS AFFILIATES, AND/OR ITS AND THEIR RESPECTIVE SUPPLIERS, MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE INFORMATION CONTAINED IN THE DOCUMENTS AND RELATED GRAPHICS, THE SOFTWARE AND SERVICES, AND OTHER MATERIAL PUBLISHED ON OR ACCESSIBLE THROUGH THIS SITE FOR ANY PURPOSE. ALL SUCH MATERIAL IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. DELINEA AND ITS AFFILIATES, AND/OR ITS AND THEIR RESPECTIVE SUPPLIERS, HEREBY DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO SUCH MATERIAL, INCLUDING ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT.

THE MATERIAL PUBLISHED ON THIS SITE COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN. DELINEA AND ITS AFFILIATES, AND/OR ITS AND THEIR RESPECTIVE SUPPLIERS, MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE MATERIAL DESCRIBED HEREIN AT ANY TIME.

Disclaimer of Liability

IN NO EVENT SHALL DELINEA AND ITS AFFILIATES, AND/OR ITS AND THEIR RESPECTIVE SUPPLIERS, BE LIABLE FOR ANY SPECIAL, INDIRECT, OR CONSEQUENTIAL DAMAGES (INCLUDING LOSS OF USE, DATA, PROFITS OR OTHER ECONOMIC ADVANTAGE) OR ANY DAMAGES WHATSOEVER, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE, OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF SOFTWARE, DOCUMENTS, PROVISION OF OR FAILURE TO PROVIDE SERVICES, OR MATERIAL AVAILABLE FROM THIS SITE.

Table of Contents

Windows API Programmer Guide	i
About this Guide	lxxvii
Intended Audience	lxxvii
Using this Guide	lxxviii
Compatibility and Limitations	lxxviii
About the Server Suite SDK	lxxix
Introduction to the Development Platform	lxxix
Windows Developer Tools	lxxix
UNIX Developer Tools	lxxx
Downloading the Standalone SDK Package	lxxxi
Installing the Standalone SDK Package	lxxxii
Overview of the Windows API Object Model	lxxxiii
How the Windows API Relies on COM Interfaces	lxxxiii
Administrative Tasks	lxxxiv
Server Suite-Specific Object Classes	lxxxiv
Creating Objects in the Proper Order	lxxxvii
Getting and Setting Object Properties	lxxxix
Interface Naming Conventions	lxxxix
Creating the Top-Level Cims Object	lxxxix
Working With NIS Maps	xc
Writing Scripts that Use API Calls	xc
Using VBScript	xc
Using PowerShell	xcii
Using Visual Studio C#	xciii
Adding Users in a One-Way Trust Environment	xcvi
Data Storage for Zones	xcvii
Classic RFC 2307 Zones (3.x, 4.x)	xcvii
Zone Attributes in Classic RFC 2307 Zones	xcvii
User Attributes in Classic RFC 2307 Zones	xcviii
Group Attributes in Classic RFC 2307 Zones	xcix
Computer Attributes in Classic RFC 2307 Zones	c
Classic Zones (2.x, 3.x, 4.x)	ci
Parent Link Attributes	ciii
Zone Attributes in Classic Zones	ciii
User Attributes in Classic Zones	civ
Group Attributes in Classic Zones	cv
Computer Attributes in Classic Zones	cvi
Hierarchical Zones (5.x)	cvii

Table of Contents

Zone Attributes in Standard Hierarchical Zones	cviii
User Attributes in Hierarchical Zones	cx
Group Attributes in Hierarchical Zones	cxii
Computer Attributes in Hierarchical Zones	cxiii
Computer-Specific Zone Attributes in Standard Hierarchical Zones	cxiii
User Attributes in RFC 2307-Compliant Zones	cxiii
Group Attributes in RFC 2307-Compliant Zones	cxiv
User Attributes in Hierarchical SFU Zones	cxiv
Group Attributes in Hierarchical SFU Zones	cxiv
The Logical Data Model for Objects	cxiv
Use of Existing Attributes	cxv
Logical Data Attributes for Zones	cxv
Logical Data Attributes for Users	cxvi
Logical Data Attributes for Groups	cxvi
Logical Data Attributes for Computers	cxvii
Logical Data Attributes for NIS Maps	cxvii
Classic SFU-Compliant Zones (version 3.5)	cxviii
Zone Attributes in Classic SFU-Compliant Zones	cxix
User Attributes in Classic SFU-Compliant Zones	cxx
Group Attributes in Classic SFU-Compliant Zones	cxxi
Classic SFU-Compliant Zones (version 4.0)	cxxi
Zone Attributes in Classic SFU 4.0 Zones	cxxi
User Attributes in Classic SFU 4.0 Zones	cxxi
Group Attributes in Classic SFU 4.0 Zones	cxxii
Using Commands and Scripts to Perform Tasks	cxxii
Getting Started with Commands and Scripts	cxxiii
Creating a Classic Zone	cxxiii
Adding a User to a Classic Zone	cxxiv
Zone Requirements	cxxv
Differences between Zone Types	cxxv
Schemas and Zones	cxxvi
How the Zone Type can Affect Features	cxxvi
Supported Zone Types	cxxvi
Server Suite Object Reference	cxxviii
AzRoleAssignment	cxxviii
Syntax	cxxviii
Methods	cxxviii
Properties	cxxix
Discussion	cxxx
GetComputerRole	cxxx
Syntax	cxxx
Return value	cxxx
Discussion	cxxx
Exceptions	cxxx

Table of Contents

Example	cxxx
Cims	cxxx
Syntax	cxxx
Discussion	cxxx
Methods	cxxx
Properties	cxxx
AddComputer	cxxx
Syntax	cxxx
Parameters	cxxx
Return value	cxxx
Exceptions	cxxx
AddComputerZone	cxxx
Syntax	cxxx
Parameters	cxxx
Return value	cxxx
Discussion	cxxx
Exceptions	cxxx
AddWindowsComputer	cxxx
Syntax	cxxx
Parameters	cxxx
Return value	cxxx
Exceptions	cxxx
ConfigureForest	cxxx
Syntax	cxxx
Parameters	cxxx
Discussion	cxxx
Exceptions	cxxx
Connect	cxxx
Syntax	cxxx
Parameters	cxxx
Discussion	cxxx
Example	cxxx
CreateZone	cxxx
Syntax	cxxx
Parameters	cxxx
Return value	cxxx
Discussion	cxxx
Exceptions	cxxx
Example	cxxx
CreateZoneWithSchema	cxxx
Syntax	cxxx
Parameters	cxxx
Return value	cxxx
Discussion	cxxx

Table of Contents

Exceptions	cxxxviii
Example	cxxxviii
GetComputer	cxxxviii
Syntax	cxxxviii
Parameter	cxxxviii
Return value	cxxxix
Discussion	cxxxix
Exceptions	cxxxix
Example	cxxxix
GetComputerByComputerZone	cxxxix
Syntax	cxxxix
Parameter	cxxxix
Return value	cxl
Discussion	cxl
Exceptions	cxl
GetComputerByPath	cxl
Syntax	cxl
Parameter	cxl
Return value	cxl
Discussion	cxl
Exceptions	cxli
Example	cxli
GetGroup	cxli
Syntax	cxli
Parameter	cxli
Return value	cxli
Discussion	cxli
Exceptions	cxlii
Example	cxlii
GetGroupByPath	cxlii
Syntax	cxlii
Parameter	cxlii
Return value	cxlii
Discussion	cxlii
Exceptions	cxliii
Example	cxliii
GetUser	cxliii
Syntax	cxliii
Parameter	cxliii
Return value	cxliii
Discussion	cxliv
Exceptions	cxliv
Example	cxliv
GetUserByPath	cxliv

Table of Contents

Syntax	cxliv
Parameter	cxliv
Return value	cxliv
Exceptions	cxliv
Discussion	cxliv
Example	cxlv
GetWindowsUser	cxlv
Syntax	cxlv
Parameter	cxlv
Return value	cxlvi
Exceptions	cxlvi
Discussion	cxlvi
GetWindowsUserByPath	cxlvi
Syntax	cxlvi
Parameter	cxlvi
Return value	cxlvi
Exceptions	cxlvi
Discussion	cxlvii
GetZone	cxlvii
Syntax	cxlvii
Parameter	cxlvii
Return value	cxlvii
Discussion	cxlvii
Exceptions	cxlvii
Example	cxlviii
GetZoneByPath	cxlviii
Syntax	cxlviii
Parameter	cxlviii
Return value	cxlviii
Discussion	cxlviii
Exceptions	cxlviii
Example	cxlix
IsForestConfigured	cxlix
Syntax	cxlix
Return value	cxlix
Exceptions	cxlix
Example	cl
LoadLicenses	cl
Syntax	cl
Return value	cl
Discussion	cl
Exceptions	cl
Example	cl
Password	cl

Table of Contents

Syntax	cl
Property value	cl
Example	cli
Server	cli
Syntax	cli
Property value	cli
Example	cli
UserName	cli
Syntax	cli
Property value	cli
Example	cli
Command	clii
Syntax	clii
Methods	clii
Properties	clii
Discussion	cli
AllowNestedExecution	cli
Syntax	cli
Property value	cli
AuthenticationType	cli
Syntax	cli
Property value	cli
CommandPattern	cli
Syntax	cli
Property value	cli
Discussion	cli
Exceptions	cli
Example	cli
CommandPatternType	cli
Syntax	cli
Property value	cli
DzdoRunAsGroupList	cli
Syntax	cli
Property value	cli
DzdoRunAsUserList	cli
Syntax	cli
Property value	cli
Discussion	cli
DzshRunAsUser	cli
Syntax	cli
Property value	cli
Guid	cli
Syntax	cli
Property value	cli

Table of Contents

IsResetVariables	clvi
Syntax	clvi
Property value	clvi
Discussion	clvi
MatchPath	clvi
Syntax	clvi
Property value	clvi
PreserveGroupMembership	clvii
Syntax	clvii
Property value	clvii
UMask	clvii
Syntax	clvii
Property value	clvii
Exceptions	clvii
VariablesToAdd	clvii
Syntax	clvii
Property value	clvii
Exceptions	clviii
VariablesToKeepOrDelete	clviii
Syntax	clviii
Property value	clviii
Discussion	clviii
Exceptions	clviii
Weight	clviii
Syntax	clviii
Property value	clviii
Discussion	clviii
Commands	clviii
Syntax	clviii
Methods	clix
GetEnumerator	clix
Syntax	clix
Return value	clix
Computer	clix
Syntax	clix
Methods	clix
Properties	clix
Commit	clx
Syntax	clx
Discussion	clx
Exceptions	clxi
Example	clxi
Delete	clxi
Syntax	clxi

Table of Contents

Discussion	clxi
Exceptions	clxi
Example	clxi
GetDirectoryEntry	clxii
Syntax	clxii
Return value	clxii
Discussion	clxii
Example	clxii
AdsilInterface	clxii
Syntax	clxii
Property value	clxii
Example	clxii
ADsPath	clxiii
Syntax	clxiii
Property value	clxiii
Example	clxiii
AgentVersion	clxiii
Syntax	clxiii
Property value	clxiii
Example	clxiii
CanonicalName	clxiv
Syntax	clxiv
Property value	clxiv
Example	clxiv
IsOrphan	clxiv
Syntax	clxiv
Property value	clxiv
Discussion	clxiv
Example	clxiv
IsReadable	clxv
Syntax	clxv
Property value	clxv
Discussion	clxv
Example	clxv
IsWritable	clxv
Syntax	clxv
Property value	clxv
Discussion	clxv
Example	clxvi
BossEnabled	clxvi
Syntax	clxvi
Property value	clxvi
Exceptions	clxvi
Example	clxvi

Table of Contents

Name	clxvi
Syntax	clxvi
Property value	clxvii
Example	clxvii
ProfileADsPath	clxvii
Syntax	clxvii
Property value	clxvii
Example	clxvii
Refresh	clxvii
Syntax	clxvii
Discussion	clxvii
Exceptions	clxviii
Example	clxviii
SchemaVersion	clxviii
Syntax	clxviii
Property value	clxviii
Discussion	clxviii
Example	clxviii
TomcatEnabled	clxix
Syntax	clxix
Property value	clxix
Exceptions	clxix
Example	clxix
Version	clxix
Syntax	clxix
Property value	clxix
Discussion	clxix
Example	clxix
WebLogicEnabled	clxx
Syntax	clxx
Property value	clxx
Exceptions	clxx
Example	clxx
WebSphereEnabled	clxx
Syntax	clxx
Property value	clxx
Exceptions	clxxi
Example	clxxi
Zone	clxxi
Syntax	clxxi
Property value	clxxi
Discussion	clxxi
Exceptions	clxxi
Example	clxxi

Table of Contents

ZoneMode	clxxii
Syntax	clxxii
Property value	clxxii
Possible values:	clxxii
ComputerGroupUnixProfiles	clxxii
Syntax	clxxii
Methods	clxxii
Properties	clxxiii
Find	clxxiii
Syntax	clxxiii
Parameter	clxxiii
Return value	clxxiii
ComputerRole	clxxiii
Syntax	clxxiii
Methods	clxxiii
Properties	clxxiv
Discussion	clxxv
AddAccessGroup	clxxv
Syntax	clxxv
Parameters	clxxvi
Discussion	clxxvi
Return value	clxxvi
Exceptions	clxxvi
AddRoleAssignment	clxxvi
Syntax	clxxvi
Return value	clxxvi
Discussion	clxxvi
AddUser	clxxvii
Syntax	clxxvii
Parameters	clxxvii
Return value	clxxvii
Discussion	clxxvii
Exceptions	clxxvii
ClearCustomAttributes	clxxvii
Syntax	clxxvii
Commit	clxxviii
Syntax	clxxviii
Discussion	clxxviii
Exceptions	clxxviii
CustomAttributes	clxxviii
Syntax	clxxviii
Property value	clxxviii
Delete	clxxviii
Syntax	clxxviii

Table of Contents

Exceptions	clxxviii
GetAccessGroup	clxxviii
Syntax	clxxviii
Parameters	clxxix
Return value	clxxix
Discussion	clxxix
Exceptions	clxxix
GetAccessGroups	clxxix
Syntax	clxxix
Return value	clxxx
GetCustomAttributeContainer	clxxx
Syntax	clxxx
Return value	clxxx
Discussion	clxxx
GetGroup	clxxx
Syntax	clxxx
Return value	clxxx
Discussion	clxxx
GetRoleAssignment	clxxx
Syntax	clxxx
Parameters	clxxx
Return value	clxxxi
Discussion	clxxxi
Exceptions	clxxxi
GetRoleAssignmentById	clxxxi
Syntax	clxxxi
Parameter	clxxxi
Return value	clxxxi
Discussion	clxxxi
Exceptions	clxxxi
GetRoleAssignments	clxxxi
Syntax	clxxxii
Return value	clxxxii
GetRoleAssignmentToAllADUsers	clxxxii
Syntax	clxxxii
Parameter	clxxxii
Return value	clxxxii
Exceptions	clxxxii
GetRoleAssignmentToEveryone	clxxxii
Syntax	clxxxii
Parameter	clxxxii
Return value	clxxxii
Discussion	clxxxiii
Exceptions	clxxxiii

Table of Contents

GetUser	clxxxiii
Syntax	clxxxiii
Parameters	clxxxiii
Return value	clxxxiii
Discussion	clxxxiii
Exceptions	clxxxiv
GetUsers	clxxxiv
Syntax	clxxxiv
Return value	clxxxiv
Validate	clxxxiv
Syntax	clxxxiv
Exceptions	clxxxiv
SetCustomAttribute	clxxxiv
Syntax	clxxxv
Parameters	clxxxv
Return value	clxxxv
Description	clxxxv
Syntax	clxxxv
Property value	clxxxv
Group	clxxxv
Syntax	clxxxv
Property value	clxxxv
IsOrphan	clxxxv
Syntax	clxxxv
Property value	clxxxv
Name	clxxxvi
Syntax	clxxxvi
Property value	clxxxvi
Zone	clxxxvi
Syntax	clxxxvi
Property value	clxxxvi
ComputerRoles	clxxxvi
Syntax	clxxxvi
Methods	clxxxvi
Properties	clxxxvi
GetEnumerator	clxxxvi
Syntax	clxxxvii
Return value	clxxxvii
IsEmpty	clxxxvii
Syntax	clxxxvii
Property value	clxxxvii
Computers	clxxxvii
Syntax	clxxxvii
Methods	clxxxvii

Table of Contents

Properties	clxxxvii
GetEnumerator	clxxxvii
Syntax	clxxxvii
Return value	clxxxviii
IsEmpty	clxxxviii
Syntax	clxxxviii
Property value	clxxxviii
ComputerUserUnixProfiles	clxxxviii
Syntax	clxxxviii
Methods	clxxxviii
Properties	clxxxviii
Find	clxxxviii
Syntax	clxxxix
Parameter	clxxxix
Return value	clxxxix
CustomAttribute	clxxxix
Properties	clxxxix
CustomAttributeContainer	clxxxix
Methods	clxxxix
GetCustomAttributes	cxc
Syntax	cxc
Parameter	cxc
Return value	cxc
ValidateCustomAttributes	cxc
Syntax	cxc
Parameter	cxc
Return value	cxc
CustomAttributes	cxc
Methods	cxc
GetEnumerator	cxc
Syntax	cxc
Return value	cxc
Entry	cxc
Syntax	cxc
Discussion	cxc
Methods	cxc
Properties	cxc
Commit	cxc
Syntax	cxc
Exceptions	cxc
Example	cxc
GetDirectoryEntry	cxc
Syntax	cxc
Return value	cxc

Table of Contents

Discussion	cxciii
Comment	cxciii
Syntax	cxciii
Property value	cxciii
Discussion	cxciii
Exceptions	cxciii
Example	cxciii
IsReadable	cxciv
Syntax	cxciv
Property value	cxciv
Discussion	cxciv
Example	cxciv
IsWritable	cxciv
Syntax	cxciv
Property value	cxciv
Discussion	cxcv
Example	cxcv
Key	cxcv
Syntax	cxcv
Property value	cxcv
Discussion	cxcv
Exceptions	cxcv
Example	cxcv
Map	cxcvi
Syntax	cxcvi
Property value	cxcvi
Value	cxcvi
Syntax	cxcvi
Property value	cxcvi
Discussion	cxcvi
Exceptions	cxcvii
Example	cxcvii
Group	cxcvii
Syntax	cxcvii
Discussion	cxcvii
Methods	cxcviii
Properties	cxcviii
AddUnixProfile	cxcviii
Syntax	cxcviii
Parameters	cxcviii
Return value	cxcix
Discussion	cxcix
Exceptions	cxcix
Example	cxcix

Table of Contents

Commit	cxcix
Syntax	cxcix
Discussion	cc
Exceptions	cc
Example	cc
CommitWithoutCheck	cc
Syntax	cc
Discussion	cc
Exceptions	cci
Example	cci
GetDirectoryEntry	cci
Syntax	cci
Return value	cci
Discussion	cci
Exceptions	cci
Example	cci
GetRoleAssignmentsFromDomain	ccii
Syntax	ccii
Parameters	ccii
Return value	ccii
Discussion	ccii
Example	ccii
GetRoleAssignmentsFromForest	cciii
Syntax	cciii
Parameters	cciii
Return value	cciii
Discussion	cciii
Example	cciii
Refresh	cciii
Syntax	cciv
Discussion	cciv
Example	cciv
AdsIInterface	cciv
Syntax	cciv
Property value	cciv
Example	cciv
ADsPath	cciv
Syntax	ccv
Property value	ccv
Example	ccv
ID	ccv
Syntax	ccv
Property value	ccv
Example	ccv

Table of Contents

UnixProfiles	ccv
Syntax	ccv
Property value	ccvi
Discussion	ccvi
Example	ccvi
GroupInfo	ccvi
Syntax	ccvi
Methods	ccvi
Properties	ccvii
Commit	ccvii
Syntax	ccvii
Delete	ccviii
Syntax	ccviii
Discussion	ccviii
Exceptions	ccviii
GetMembers	ccviii
Syntax	ccviii
Return value	ccviii
Discussion	ccviii
Import	ccviii
Syntax	ccviii
Parameter	ccviii
Discussion	ccix
UpdateStatus	ccix
Syntax	ccix
Discussion	ccix
CandidateDN	ccix
Syntax	ccix
Property value	ccix
Discussion	ccix
GID	ccix
Syntax	ccix
Property value	ccx
Discussion	ccx
ID	ccx
Syntax	ccx
Property value	ccx
IsImported	ccx
Syntax	ccx
Property value	ccx
Discussion	ccx
Members	ccx
Syntax	ccx
Property value	ccx

Table of Contents

Name	ccxi
Syntax	ccxi
Property value	ccxi
Source	ccxi
Syntax	ccxi
Property value	ccxi
Discussion	ccxi
Status	ccxi
Syntax	ccxi
Property value	ccxi
Discussion	ccxi
StatusDescription	ccxii
Syntax	ccxii
Property value	ccxii
Discussion	ccxii
TimeStamp	ccxii
Syntax	ccxii
Property value	ccxii
Example	ccxii
GroupInfos	ccxiii
Syntax	ccxiii
Methods	ccxiii
Properties	ccxiii
Find	ccxiii
Syntax	ccxiii
Parameter	ccxiv
Return value	ccxiv
GetEnumerator	ccxiv
Syntax	ccxiv
Return value	ccxiv
Count	ccxiv
Syntax	ccxiv
Property value	ccxiv
Discussion	ccxiv
IsEmpty	ccxiv
Syntax	ccxiv
Property value	ccxv
Discussion	ccxv
GroupMember	ccxv
Syntax	ccxv
Methods	ccxv
Properties	ccxv
CandidateDN	ccxv
Syntax	ccxv

Table of Contents

Property value	ccxvi
Discussion	ccxvi
Name	ccxvi
Syntax	ccxvi
Property value	ccxvi
GroupMembers	ccxvi
Syntax	ccxvi
Methods	ccxvi
Properties	ccxvii
Add	ccxvii
Syntax	ccxvii
Parameter	ccxvii
Return value	ccxvii
AddRange	ccxvii
Syntax	ccxvii
Parameter	ccxvii
Clear	ccxvii
Syntax	ccxvii
Discussion	ccxviii
Remove	ccxviii
Syntax	ccxviii
Parameter	ccxviii
Count	ccxviii
Syntax	ccxviii
Property value	ccxviii
Discussion	ccxviii
GroupUnixProfile	ccxviii
Syntax	ccxviii
Discussion	ccxviii
Methods	ccxix
Properties	ccxix
Commit	ccxx
Syntax	ccxx
Discussion	ccxx
Exceptions	ccxx
Example	ccxx
Delete	ccxxi
Syntax	ccxxi
Discussion	ccxxi
Example	ccxxi
GetDirectoryEntry	ccxxi
Syntax	ccxxi
Return value	ccxxi
Discussion	ccxxi

Table of Contents

Example	ccxxi
Refresh	ccxxii
Syntax	ccxxii
Discussion	ccxxii
Example	ccxxii
Validate	ccxxii
Syntax	ccxxii
Discussion	ccxxii
Exceptions	ccxxiii
Example	ccxxiii
ADsPath	ccxxiii
Syntax	ccxxiii
Property value	ccxxiii
Example	ccxxiii
Cims	ccxxiii
Syntax	ccxxiv
Property value	ccxxiv
Discussion	ccxxiv
Example	ccxxiv
Group	ccxxiv
Syntax	ccxxiv
Property value	ccxxiv
Example	ccxxiv
GroupID	ccxxv
Syntax	ccxxv
Property value	ccxxv
Discussion	ccxxv
Exceptions	ccxxv
ID	ccxxv
Syntax	ccxxv
Property value	ccxxv
Example	ccxxv
IsForeign	ccxxv
Syntax	ccxxvi
Property value	ccxxvi
Discussion	ccxxvi
Example	ccxxvi
IsMembershipRequired	ccxxvi
Syntax	ccxxvi
Property value	ccxxvi
Discussion	ccxxvi
Exceptions	ccxxvi
Example	ccxxvii
IsOrphan	ccxxvii

Table of Contents

Syntax	ccxxvii
Property value	ccxxvii
Discussion	ccxxvii
Exceptions	ccxxvii
Example	ccxxvii
IsReadable	ccxxvii
Syntax	ccxxvii
Property value	ccxxviii
Discussion	ccxxviii
Example	ccxxviii
IsSFU	ccxxviii
Syntax	ccxxviii
Property value	ccxxviii
Discussion	ccxxviii
IsWritable	ccxxviii
Syntax	ccxxviii
Property value	ccxxix
Discussion	ccxxix
Example	ccxxix
Members	ccxxix
Syntax	ccxxix
Property value	ccxxix
Exceptions	ccxxix
ProfileState	ccxxix
Syntax	ccxxix
Property value	ccxxx
Exceptions	ccxxx
Name	ccxxx
Syntax	ccxxx
Property value	ccxxx
Example	ccxxx
Type	ccxxx
Syntax	ccxxx
Property value	ccxxx
Discussion	ccxxx
Example	ccxxx
UnixEnabled	ccxxx
Syntax	ccxxx
Property value	ccxxx
Example	ccxxx
Zone	ccxxx
Syntax	ccxxx
Property value	ccxxx
Discussion	ccxxx

Table of Contents

GroupUnixProfiles	ccxxxii
Syntax	ccxxxii
Discussion	ccxxxii
Methods	ccxxxii
Properties	ccxxxii
GetEnumerator	ccxxxii
Syntax	ccxxxiii
Return value	ccxxxiii
Refresh	ccxxxiii
Syntax	ccxxxiii
Discussion	ccxxxiii
Count	ccxxxiii
Syntax	ccxxxiii
Property value	ccxxxiii
Discussion	ccxxxiii
Example	ccxxxiii
IsEmpty	ccxxxiv
Syntax	ccxxxiv
Property value	ccxxxiv
Discussion	ccxxxiv
Example	ccxxxiv
HierarchicalGroup	ccxxxiv
Syntax	ccxxxiv
Methods	ccxxxiv
Properties	ccxxxv
GetComputer	ccxxxvii
Syntax	ccxxxvii
Return value	ccxxxvii
InheritFromParent	ccxxxviii
Syntax	ccxxxviii
Discussion	ccxxxviii
ResolveEffectiveProfile	ccxxxviii
Syntax	ccxxxviii
Discussion	ccxxxviii
EffectiveGid	ccxxxviii
Syntax	ccxxxviii
Property value	ccxxxviii
Exceptions	ccxxxviii
EffectiveMembers	ccxxxviii
Syntax	ccxxxviii
Property value	ccxxxix
Exceptions	ccxxxix
EffectiveIsMembershipRequired	ccxxxix
Syntax	ccxxxix

Table of Contents

Property value	ccxxxix
Discussion	ccxxxix
Exceptions	ccxxxix
EffectiveName	ccxxxix
Syntax	ccxxxix
Property value	ccxxxix
EffectiveProfileState	ccxxxix
Syntax	ccxl
Property value	ccxl
Exceptions	ccxl
IsEffectiveGidDefined	ccxl
Syntax	ccxl
Property value	ccxl
Discussion	ccxl
IsEffectiveIsMembershipRequiredDefined	ccxl
Syntax	ccxl
Property value	ccxl
Discussion	ccxl
IsEffectiveMembersDefined	ccxl
Syntax	ccxl
Property value	ccxli
Discussion	ccxli
Exceptions	ccxli
IsEffectiveNameDefined	ccxli
Syntax	ccxli
Property value	ccxli
Discussion	ccxli
IsEffectiveProfileStateDefined	ccxli
Syntax	ccxli
Property value	ccxli
Discussion	ccxli
Exceptions	ccxlii
IsGidDefined	ccxlii
Syntax	ccxlii
Property value	ccxlii
Exceptions	ccxlii
IsMembersDefined	ccxlii
Syntax	ccxlii
Property value	ccxlii
Exceptions	ccxlii
IsMembershipRequiredDefined	ccxlii
Syntax	ccxlii
Property value	ccxlii
Exceptions	ccxlili

Table of Contents

IsNameDefined	ccxliii
Syntax	ccxliii
Property value	ccxliii
Exceptions	ccxliii
IsProfileStateDefined	ccxliii
Syntax	ccxliii
Property value	ccxliii
Discussion	ccxliii
Exceptions	ccxliii
Zone	ccxliii
Syntax	ccxliii
Property value	ccxliv
HierarchicalUser	ccxliv
Syntax	ccxliv
Discussion	ccxliv
Methods	ccxliv
Properties	ccxlv
AddUserRoleAssignment	ccxlviii
Syntax	ccxlviii
Return value	ccxlix
Discussion	ccxlix
Example	ccxlix
GetComputer	ccxlix
Syntax	ccxlix
Return value	ccxlix
GetEffectiveUserRoleAssignments	ccl
Syntax	ccl
Return value	ccl
Discussion	ccl
GetUserRoleAssignment	ccl
Syntax	ccl
Parameter	ccl
Return value	ccl
Exceptions	ccl
Example	ccl
GetUserRoleAssignments	ccli
Syntax	ccli
Return value	ccli
Discussion	ccli
InheritFromParent	ccli
Syntax	ccli
Discussion	ccli
ResolveEffectiveProfile	ccli
Syntax	ccli

Table of Contents

Discussion	cclii
ResolveEffectiveRoles	cclii
Syntax	cclii
Parameters	cclii
Discussion	cclii
EffectiveGecos	cclii
Syntax	ccliii
Property value	ccliii
Discussion	ccliii
EffectiveGecosZone	ccliii
Syntax	ccliii
Property value	ccliii
Discussion	ccliii
EffectiveHomeDirectory	ccliii
Syntax	ccliii
Property value	ccliii
Discussion	ccliii
EffectiveHomeDirectoryZone	ccliii
Syntax	ccliv
Property value	ccliv
Discussion	ccliv
EffectiveIsUseAutoPrivateGroup	ccliv
Syntax	ccliv
Property value	ccliv
Discussion	ccliv
EffectiveName	ccliv
Syntax	ccliv
Property value	ccliv
Discussion	ccliv
EffectiveNameZone	ccliv
Syntax	cclv
Property value	cclv
Discussion	cclv
EffectivePrimaryGroup	cclv
Syntax	cclv
Property value	cclv
Discussion	cclv
EffectiveProfileState	cclv
Syntax	cclv
Property value	cclv
Discussion	cclv
Exceptions	cclv
EffectiveProfileStateZone	cclvi
Syntax	cclvi

Table of Contents

Property value	cclvi
Discussion	cclvi
Exceptions	cclvi
EffectivePrimaryGroupZone	cclvi
Syntax	cclvi
Property value	cclvi
Discussion	cclvi
EffectiveShell	cclvi
Syntax	cclvi
Property value	cclvi
Discussion	cclvii
EffectiveShellZone	cclvii
Syntax	cclvii
Property value	cclvii
Discussion	cclvii
EffectiveUid	cclvii
Syntax	cclvii
Property value	cclvii
Discussion	cclvii
EffectiveUidZone	cclvii
Syntax	cclvii
Property value	cclvii
Discussion	cclviii
Gecos	cclviii
Syntax	cclviii
Property value	cclviii
Discussion	cclviii
IsEffectiveGecosDefined	cclviii
Syntax	cclviii
Property value	cclviii
Discussion	cclviii
IsEffectiveHomeDirectoryDefined	cclviii
Syntax	cclviii
Property value	cclviii
Discussion	cclviii
IsEffectiveNameDefined	cclix
Syntax	cclix
Property value	cclix
Discussion	cclix
IsEffectivePrimaryGroupDefined	cclix
Syntax	cclix
Property value	cclix
Discussion	cclix
IsEffectiveProfileStateDefined	cclix

Table of Contents

Syntax	cclix
Property value	cclix
Exceptions	cclix
IsEffectiveShellDefined	cclix
Syntax	cclx
Property value	cclx
Discussion	cclx
IsEffectiveUidDefined	cclx
Syntax	cclx
Property value	cclx
Discussion	cclx
IsEffectiveUseAutoPrivateGroupDefined	cclx
Syntax	cclx
Property value	cclx
Discussion	cclx
IsGecosDefined	cclx
Syntax	cclx
Property value	cclxi
Exceptions	cclxi
IsHomeDirectoryDefined	cclxi
Syntax	cclxi
Property value	cclxi
Exceptions	cclxi
IsNameDefined	cclxi
Syntax	cclxi
Property value	cclxi
Exceptions	cclxi
IsProfileStateDefined	cclxi
Syntax	cclxi
Property value	cclxi
Exceptions	cclxii
IsPrimaryGroupDefined	cclxii
Syntax	cclxii
Property value	cclxii
Discussion	cclxii
Exceptions	cclxii
IsSecondary	cclxii
Syntax	cclxii
Property value	cclxii
IsShellDefined	cclxii
Syntax	cclxii
Property value	cclxii
Exceptions	cclxiii
IsUidDefined	cclxiii

Table of Contents

Syntax	cclxiii
Property value	cclxiii
Exceptions	cclxiii
IsUseAutoPrivateGroup	cclxiii
Syntax	cclxiii
Property value	cclxiii
Discussion	cclxiii
IsUseAutoPrivateGroupDefined	cclxiii
Syntax	cclxiii
Property value	cclxiii
Discussion	cclxiv
Exceptions	cclxiv
Zone	cclxiv
Syntax	cclxiv
Property value	cclxiv
HierarchicalZone	cclxiv
Syntax	cclxiv
Discussion	cclxiv
Methods	cclxiv
Properties	cclxx
AddAccessGroup	cclxxiii
Syntax	cclxxiii
Parameters	cclxxiv
Return value	cclxxiv
Discussion	cclxxiv
Exceptions	cclxxiv
Example	cclxxiv
AddComputerRole	cclxxv
Syntax	cclxxv
Parameters	cclxxv
Return value	cclxxv
AddGroupPartialProfile	cclxxv
Syntax	cclxxv
Parameters	cclxxv
Return value	cclxxvi
Discussion	cclxxvi
Exceptions	cclxxvi
Example	cclxxvi
AddLocalGroupPartialProfile	cclxxvi
Syntax	cclxxvii
Parameters	cclxxvii
Return value	cclxxvii
Exceptions	cclxxvii
AddRoleAssignment	cclxxvii

Table of Contents

Syntax	cclxxvii
Return value	cclxxvii
AddLocalUserPartialProfile	cclxxvii
Syntax	cclxxvii
Parameters	cclxxvii
Return value	cclxxvii
Exceptions	cclxxvii
AddUserPartialProfile	cclxxvii
Syntax	cclxxviii
Parameters	cclxxviii
Return value	cclxxviii
Discussion	cclxxviii
Exceptions	cclxxviii
Example	cclxxviii
CreateCommand	cclxxix
Syntax	cclxxix
Return value	cclxxix
Discussion	cclxxix
Example	cclxxix
CreateNetworkAccess	cclxxx
Syntax	cclxxx
Return value	cclxxx
Discussion	cclxxx
Example	cclxxx
CreatePamAccess	cclxxxix
Syntax	cclxxxix
Return value	cclxxxix
Discussion	cclxxxix
Example	cclxxxix
CreateRole	cclxxxix
Syntax	cclxxxix
Parameter	cclxxxix
Return value	cclxxxix
Discussion	cclxxxix
Example	cclxxxix
CreateSshRight	cclxxxix
Syntax	cclxxxix
Return value	cclxxxix
Discussion	cclxxxix
CreateWindowsApplication	cclxxxix
Syntax	cclxxxix
Return value	cclxxxix
Discussion	cclxxxix
Example	cclxxxix

Table of Contents

CreateWindowsDesktop	cclxxxiv
Syntax	cclxxxiv
Return value	cclxxxiv
Discussion	cclxxxiv
Example	cclxxxiv
GeneratePredefinedRights	cclxxxv
Syntax	cclxxxv
Discussion	cclxxxv
GeneratePredefinedRoles	cclxxxv
Syntax	cclxxxv
Discussion	cclxxxv
GetAccessGroup	cclxxxv
Syntax	cclxxxvi
Parameters	cclxxxvi
Return value	cclxxxvi
Discussion	cclxxxvi
Exceptions	cclxxxvi
Example	cclxxxvii
GetAccessGroups	cclxxxvii
Syntax	cclxxxvii
Return value	cclxxxvii
GetChildZones	cclxxxvii
Syntax	cclxxxvii
Return value	cclxxxvii
Exceptions	cclxxxvii
GetCommand	cclxxxvii
Syntax	cclxxxvii
Parameter	cclxxxvii
Return value	cclxxxviii
Exceptions	cclxxxviii
Example	cclxxxviii
GetCommands	cclxxxviii
Syntax	cclxxxviii
Return value	cclxxxviii
GetComputerRole	cclxxxix
Syntax	cclxxxix
Parameter	cclxxxix
Return value	cclxxxix
Exceptions	cclxxxix
Example	cclxxxix
GetComputerRoles	ccxc
Syntax	ccxc
Return value	ccxc
GetEffectiveCommands	ccxc

Table of Contents

Syntax	CCXC
Return value	CCXC
Exceptions	CCXC
GetEffectiveNetworkAccesses	CCXC
Syntax	CCXC
Return value	CCXC
Exceptions	CCXC
GetEffectivePamAccesses	CCXci
Syntax	CCXci
Return value	CCXci
Exceptions	CCXci
GetEffectiveRoles	CCXci
Syntax	CCXci
Return value	CCXci
Exceptions	CCXci
GetEffectiveSshs	CCXci
Syntax	CCXci
Return value	CCXci
Exceptions	CCXci
GetEffectiveUserUnixProfiles	CCXcii
Syntax	CCXcii
Return value	CCXcii
GetEffectiveWindowsApplications	CCXcii
Syntax	CCXcii
Return value	CCXcii
Exceptions	CCXcii
GetEffectiveWindowsDesktops	CCXcii
Syntax	CCXcii
Return value	CCXcii
Exceptions	CCXcii
GetEffectiveWindowsUsers	CCXcii
Syntax	CCXciii
Return value	CCXciii
GetNetworkAccess	CCXciii
Syntax	CCXciii
Parameter	CCXciii
Return value	CCXciii
Exceptions	CCXciii
Example	CCXciii
GetNetworkAccesses	CCXciii
Syntax	CCXciii
Return value	CCXciii
Discussion	CCXciv
GetNSSVariable	CCXciv

Table of Contents

Syntax	ccxciv
Parameter	ccxciv
Return value	ccxciv
GetNSSVariables	ccxciv
Syntax	ccxciv
Return value	ccxciv
GetPamAccess	ccxciv
Syntax	ccxciv
Parameter	ccxciv
Return value	ccxcv
Exceptions	ccxcv
Example	ccxcv
GetPamAccesses	ccxcv
Syntax	ccxcv
Return value	ccxcv
GetPrimaryUser	ccxcv
Syntax	ccxcv
Parameters	ccxcv
Return value	ccxcvi
Discussion	ccxcvi
Exceptions	ccxcvi
GetRole	ccxcvi
Syntax	ccxcvi
Parameter	ccxcvi
Return value	ccxcvi
Exceptions	ccxcvii
Example	ccxcvii
GetRoleAssignment	ccxcvii
Syntax	ccxcvii
Parameter	ccxcvii
Return value	ccxcviii
Exceptions	ccxcviii
GetRoleAssignmentById	ccxcviii
Syntax	ccxcviii
Parameter	ccxcviii
Return value	ccxcviii
Exceptions	ccxcviii
GetRoleAssignments	ccxcviii
Syntax	ccxcviii
Return value	ccxcviii
GetRoleAssignmentToAllADUsers	ccxcix
Syntax	ccxcix
Parameter	ccxcix
Return value	ccxcix

Table of Contents

Exceptions	ccxcix
GetRoleAssignmentToAllUnixUsers	ccxcix
Syntax	ccxcix
Parameter	ccxcix
Return value	ccxcix
Discussion	ccxcix
Exceptions	ccxcix
GetRoles	ccc
Syntax	ccc
Return value	ccc
GetSecondaryUsers	ccc
Syntax	ccc
Parameters	ccc
Return value	ccc
Discussion	ccc
Exceptions	ccc
GetSshRight	ccci
Syntax	ccci
Parameter	ccci
Return value	ccci
Exceptions	ccci
GetSshRights	ccci
Syntax	ccci
Return value	ccci
GetSubTreeRoleAssignments	ccci
Syntax	ccci
Return value	ccci
Exceptions	cccii
GetUserProfiles	cccii
Syntax	cccii
Parameters	cccii
Return value	cccii
Exceptions	cccii
GetUserRoleAssignments	cccii
Syntax	cccii
Parameters	ccciii
Return value	ccciii
Exceptions	ccciii
GetWindowsApplication	ccciii
Syntax	ccciii
Parameter	ccciii
Return value	ccciii
Exceptions	ccciii
GetWindowsApplications	ccciv

Table of Contents

Syntax	ccciv
Return value	ccciv
GetWindowsComputers	ccciv
Syntax	ccciv
Return value	ccciv
GetWindowsDesktop	ccciv
Syntax	ccciv
Parameter	ccciv
Return value	ccciv
Exceptions	ccciv
GetWindowsDesktops	ccciv
Syntax	ccciv
Return value	ccciv
PrecreateComputerZone	ccciv
Syntax	ccciv
Parameters	ccciv
Return value	ccciv
Discussion	cccvi
Exceptions	cccvi
SetNSSVariable	cccvi
Syntax	cccvi
Parameter	cccvi
GroupDefaultName	cccvi
Syntax	cccvi
Property value	cccvi
IsChild	cccvi
Syntax	cccvii
Property value	cccvii
Discussion	cccvii
IsGroupDefaultNameDefined	cccvii
Syntax	cccvii
Property value	cccvii
Exceptions	cccvii
IsNextGidDefined	cccvii
Syntax	cccvii
Property value	cccvii
Exceptions	cccvii
IsNextUidDefined	cccviii
Syntax	cccviii
Property value	cccviii
Exceptions	cccviii
IsUseAutoPrivateGroupDefined	cccviii
Syntax	cccviii
Property value	cccviii

Table of Contents

Discussion	cccviii
Exceptions	cccviii
IsUserDefaultGecosDefined	cccviii
Syntax	cccviii
Property value	cccviii
Exceptions	cccix
IsUserDefaultHomeDirectoryDefined	cccix
Syntax	cccix
Property value	cccix
Exceptions	cccix
IsUserDefaultNameDefined	cccix
Syntax	cccix
Property value	cccix
Exceptions	cccix
IsUserDefaultPrimaryGroupDefined	cccix
Syntax	cccix
Property value	cccix
Exceptions	cccix
IsUserDefaultRoleDefined	cccix
Syntax	cccix
Property value	cccix
Exceptions	cccix
IsUserDefaultShellDefined	cccix
Syntax	cccix
Property value	cccix
Exceptions	cccix
NssVariables	cccix
Syntax	cccix
Property value	cccix
Discussion	cccxi
Example	cccxi
Parent	cccxi
Syntax	cccxi
Property value	cccxi
Discussion	cccxi
Exceptions	cccxi
Example	cccxi
UseAppleGid	cccxi
Syntax	cccxi
Property value	cccxi
UseAppleUid	cccxi
Syntax	cccxi
Property value	cccxi
UseAutoGid	cccxi

Table of Contents

Syntax	cccxi
Property value	cccxi
UseAutoPrivateGroup	cccxi
Syntax	cccxi
Property value	cccxi
Discussion	cccxi
Exceptions	cccxi
UseAutoUid	cccxi
Syntax	cccxi
Property value	cccxi
UseNextGid	cccxi
Syntax	cccxi
Property value	cccxi
Discussion	cccxi
Example	cccxi
UseNextUid	cccxi
Syntax	cccxi
Property value	cccxi
Discussion	cccxi
Example	cccxi
UserDefaultGecos	cccxi
Syntax	cccxi
Property value	cccxi
Example	cccxi
UserDefaultGid	cccxi
Syntax	cccxi
Property value	cccxi
Discussion	cccxi
Exceptions	cccxi
UserDefaultName	cccxi
Syntax	cccxi
Property value	cccxi
UserDefaultPrimaryGroup	cccxi
Syntax	cccxi
Property value	cccxi
Discussion	cccxi
Exceptions	cccxi
UserDefaultRole	cccxi
Syntax	cccxi
Property value	cccxi
HierarchicalZoneComputer	cccxi
Syntax	cccxi
Discussion	cccxi
Methods	cccxi

Table of Contents

Properties	cccxxii
AddAccessGroup	cccxxiii
Syntax	cccxxiii
Parameters	cccxxiv
Return value	cccxxiv
Discussion	cccxxiv
Exceptions	cccxxiv
AddGroupPartialProfile	cccxxiv
Syntax	cccxxiv
Parameters	cccxxiv
Return value	cccxxv
Discussion	cccxxv
Exceptions	cccxxv
Example	cccxxv
AddRoleAssignment	cccxxv
Syntax	cccxxv
Return value	cccxxv
AddLocalGroupPartialProfile	cccxxv
Syntax	cccxxvi
Parameters	cccxxvi
Return value	cccxxvi
Exceptions	cccxxvi
AddLocalUserPartialProfile	cccxxvi
Syntax	cccxxvi
Parameters	cccxxvi
Return value	cccxxvi
Exceptions	cccxxvi
AddUserPartialProfile	cccxxvi
Syntax	cccxxvi
Parameters	cccxxvi
Return value	cccxxvii
Discussion	cccxxvii
Exceptions	cccxxvii
Example	cccxxvii
ComputerZoneADsPath	cccxxvii
Syntax	cccxxvii
Property value	cccxxvii
Discussion	cccxxviii
CreateImportPendingGroup	cccxxviii
Syntax	cccxxviii
Parameters	cccxxviii
Return value	cccxxviii
Discussion	cccxxviii
CreateImportPendingUser	cccxxviii

Table of Contents

Syntax	cccxxviii
Parameters	cccxxviii
Return value	cccxxix
Discussion	cccxxix
DeleteAllProfiles	cccxxix
Syntax	cccxxix
Discussion	cccxxix
DeleteZone	cccxxix
Syntax	cccxxix
Discussion	cccxxix
GetAccessGroup	cccxxix
Syntax	cccxxx
Parameters	cccxxx
Return value	cccxxx
Discussion	cccxxx
Exceptions	cccxxx
Example	cccxxxi
GetAccessGroups	cccxxxi
Syntax	cccxxxi
Return value	cccxxxi
GetEffectiveUserUnixProfiles	cccxxxi
Syntax	cccxxxi
Return value	cccxxxi
GetGroupUnixProfile	cccxxxi
Syntax	cccxxxi
Parameter	cccxxxi
Return value	cccxxxi
Discussion	cccxxxii
Exceptions	cccxxxii
GetGroupUnixProfileByDN	cccxxxii
Syntax	cccxxxii
Parameter	cccxxxii
Return value	cccxxxii
Discussion	cccxxxii
Exceptions	cccxxxii
GetGroupUnixProfileByName	cccxxxii
Syntax	cccxxxii
Parameter	cccxxxiii
Return value	cccxxxiii
Discussion	cccxxxiii
Exceptions	cccxxxiii
GetGroupUnixProfiles	cccxxxiii
Syntax	cccxxxiii
Return value	cccxxxiii

Table of Contents

GetImportPendingGroup	cccxxxiii
Syntax	cccxxxiii
Parameter	cccxxxiii
Return value	cccxxxiv
Discussion	cccxxxiv
GetImportPendingGroups	cccxxxiv
Syntax	cccxxxiv
Return value	cccxxxiv
GetImportPendingUser	cccxxxiv
Syntax	cccxxxiv
Parameter	cccxxxiv
Return value	cccxxxv
Discussion	cccxxxv
GetImportPendingUsers	cccxxxv
Syntax	cccxxxv
Return value	cccxxxv
GetIPendingGroupID	cccxxxv
Syntax	cccxxxv
Return value	cccxxxv
GetIPendingUserID	cccxxxv
Syntax	cccxxxv
Return value	cccxxxvi
GetLocalGroupUnixProfile	cccxxxvi
Syntax	cccxxxvi
Parameter	cccxxxvi
Return value	cccxxxvi
Exceptions	cccxxxvi
GetLocalGroupUnixProfileByDN	cccxxxvi
Syntax	cccxxxvi
Parameter	cccxxxvi
Return value	cccxxxvi
GetLocalGroupUnixProfileByGid (Int32)	cccxxxvii
Syntax	cccxxxvii
Parameter	cccxxxvii
Return value	cccxxxvii
GetLocalGroupUnixProfiles	cccxxxvii
Syntax	cccxxxvii
Return value	cccxxxvii
GetLocalUserUnixProfile	cccxxxvii
Syntax	cccxxxvii
Parameter	cccxxxvii
Return value	cccxxxvii
GetLocalUserUnixProfileByDN	cccxxxvii
Syntax	cccxxxviii

Table of Contents

Parameter	cccxviii
GetLocalUserUnixProfileByUid (Int32)	cccxviii
Syntax	cccxviii
Parameter	cccxviii
Return value	cccxviii
GetLocalUserUnixProfiles	cccxviii
Syntax	cccxviii
Return value	cccxviii
GetNssVariable	cccxviii
Syntax	cccxviii
Parameter	cccxix
Return value	cccxix
GetNSSVariables	cccxix
Syntax	cccxix
Return value	cccxix
GetPrimaryUser	cccxix
Syntax	cccxix
Parameters	cccxix
Return value	cccxl
Discussion	cccxl
Exceptions	cccxl
GetRoleAssignment	cccxl
Syntax	cccxl
Parameter	cccxl
Return value	cccxl
Exceptions	cccxl
GetRoleAssignmentById	cccxl
Syntax	cccxl
Parameter	cccxl
Return value	cccxli
Exceptions	cccxli
GetRoleAssignments	cccxli
Syntax	cccxli
Return value	cccxli
GetRoleAssignmentToAllADUsers	cccxli
Syntax	cccxli
Parameter	cccxli
Return value	cccxli
Exceptions	cccxli
GetRoleAssignmentToAllUnixUsers	cccxli
Syntax	cccxlii
Parameter	cccxlii
Return value	cccxlii
Discussion	cccxlii

Table of Contents

Exceptions	cccxlvi
GetSecondaryUsers	cccxlvi
Syntax	cccxlvi
Parameters	cccxlvi
Return value	cccxlvi
Discussion	cccxlvi
Exceptions	cccxlvi
GetUserProfiles	cccxlvi
Syntax	cccxlvi
Parameters	cccxlvi
Return value	cccxlvi
Discussion	cccxlvi
Exceptions	cccxlvi
GetUserRoleAssignments	cccxlvi
Syntax	cccxlvi
Parameters	cccxlvi
Return value	cccxlvi
Discussion	cccxlvi
Exceptions	cccxlvi
GetUserUnixProfile	cccxlvi
Syntax	cccxlvi
Parameter	cccxlvi
Return value	cccxlvi
Discussion	cccxlvi
Exceptions	cccxlvi
GetUserUnixProfileByDN	cccxlvi
Syntax	cccxlvi
Parameter	cccxlvi
Return value	cccxlvi
Discussion	cccxlvi
Exceptions	cccxlvi
GetUserUnixProfileByName	cccxlvi
Syntax	cccxlvi
Parameter	cccxlvi
Return value	cccxlvi
Exceptions	cccxlvi
GetUserUnixProfileByUid	cccxlvi
Syntax	cccxlvi
Parameter	cccxlvi
Return value	cccxlvi
Discussion	cccxlvi
Exceptions	cccxlvi
GetUserUnixProfiles	cccxlvi
Syntax	cccxlvi

Table of Contents

Return value	cccxlvii
GroupUnixProfileExists	cccxlvii
Syntax	cccxlvii
Parameter	cccxlvii
Return value	cccxlviii
Exceptions	cccxlviii
IsOrphanZone	cccxlviii
Syntax	cccxlviii
Property value	cccxlviii
Discussion	cccxlviii
LocalGroupUnixProfileExists	cccxlviii
Syntax	cccxlviii
Parameter	cccxlviii
Return value	cccclix
Exceptions	cccclix
LocalUserUnixProfileExists	cccclix
Syntax	cccclix
Parameter	cccclix
Return value	cccclix
Exceptions	cccclix
NssVariables	cccclix
Syntax	cccclix
Property value	ccccl
Discussion	ccccl
SetNSSVariable	ccccl
Syntax	ccccl
Parameter	ccccl
UserHomeDirectory	ccccl
Syntax	ccccl
Property value	ccccl
UserShell	ccccl
Syntax	ccccl
Property value	ccccl
UserUnixProfileExists	ccccl
Syntax	ccccli
Parameter	ccccli
Return value	ccccli
Exceptions	ccccli
Zone	ccccli
Syntax	ccccli
Property value	ccccli
Discussion	ccccli
Exceptions	ccccli
HzRoleAssignment	ccccli

Table of Contents

Syntax	ccclii
Methods	ccclii
Properties	ccclii
Zone	cccliii
Syntax	cccliii
Property value	cccliii
InheritedRoleAsg	cccliii
Syntax	cccliii
Methods	cccliii
Properties	cccliii
GetTrustee	cccliv
Syntax	cccliv
Return value	cccliv
EndTime	cccliv
Syntax	cccliv
Property value	cccliv
IsRoleOrphaned	cccliv
Syntax	cccliv
Property value	cccliv
IsTrusteeOrphaned	ccclv
Syntax	ccclv
Property value	ccclv
Role	ccclv
Syntax	ccclv
Property value	ccclv
Source	ccclv
Syntax	ccclv
Property value	ccclv
StartTime	ccclv
Syntax	ccclv
Property value	ccclv
TrusteeDn	ccclv
Syntax	ccclv
Property value	ccclvi
Key	ccclvi
Syntax	ccclvi
Discussion	ccclvi
Properties	ccclvi
Count	ccclvi
Syntax	ccclvi
Property value	ccclvi
Discussion	ccclvii
Example	ccclvii
ExpiryDate	ccclvii

Table of Contents

Syntax	ccclvii
Property value	ccclvii
Discussion	ccclvii
Example	ccclvii
IsEval	ccclviii
Syntax	ccclviii
Property value	ccclviii
Discussion	ccclviii
Example	ccclviii
IsValid	ccclix
Syntax	ccclix
Property value	ccclix
Discussion	ccclix
Example	ccclix
SerialNumber	ccclix
Syntax	ccclix
Property value	ccclix
Discussion	ccclix
Example	ccclix
Type	ccclix
Syntax	ccclix
Property value	ccclix
Discussion	ccclix
Example	ccclix
Keys	ccclxi
Syntax	ccclxi
Discussion	ccclxi
Methods	ccclxi
Properties	ccclxi
Add	ccclxii
Syntax	ccclxii
Parameter	ccclxii
Return value	ccclxii
Exceptions	ccclxii
GetEnumerator	ccclxii
Syntax	ccclxii
Return value	ccclxii
Remove	ccclxii
Syntax	ccclxii
Parameter	ccclxii
Return value	ccclxiii
Exceptions	ccclxiii
Count	ccclxiii
Syntax	ccclxiii

Table of Contents

Property value	ccclxiii
Item	ccclxiii
Syntax	ccclxiii
Parameter	ccclxiii
Property value	ccclxiii
License	ccclxiii
Syntax	ccclxiv
Discussion	ccclxiv
Properties	ccclxiv
Count	ccclxiv
Syntax	ccclxiv
Property value	ccclxiv
Example	ccclxiv
IsEval	ccclxv
Syntax	ccclxv
Property value	ccclxv
Discussion	ccclxv
Keys	ccclxv
Syntax	ccclxv
Property value	ccclxv
Type	ccclxv
Syntax	ccclxv
Property value	ccclxv
Discussion	ccclxvi
Example	ccclxvi
UsedCount	ccclxvi
Syntax	ccclxvi
Property value	ccclxvi
Discussion	ccclxvi
Exceptions	ccclxvii
Example	ccclxvii
Licenses	ccclxvii
Syntax	ccclxvii
Discussion	ccclxvii
Methods	ccclxvii
Properties	ccclxvii
AddLicenseKey	ccclxviii
Syntax	ccclxviii
Parameter	ccclxviii
Return value	ccclxviii
Exceptions	ccclxviii
Commit	ccclxviii
Syntax	ccclxix
Discussion	ccclxix

Table of Contents

Exceptions	ccclxix
GetDirectoryEntry	ccclxix
Syntax	ccclxix
Return value	ccclxix
Discussion	ccclxix
Refresh	ccclxix
Syntax	ccclxix
Discussion	ccclxix
RemoveLicenseKey	ccclxix
Syntax	ccclxix
Parameter	ccclxix
Return value	ccclxx
Exceptions	ccclxx
Count	ccclxx
Syntax	ccclxx
Property value	ccclxx
HasEvaluation	ccclxx
Syntax	ccclxx
Property value	ccclxx
HasMachineLicense	ccclxx
Syntax	ccclxx
Property value	ccclxx
ID	ccclxxi
Syntax	ccclxxi
Property value	ccclxxi
IsReadable	ccclxxi
Syntax	ccclxxi
Property value	ccclxxi
Discussion	ccclxxi
IsWritable	ccclxxi
Syntax	ccclxxi
Property value	ccclxxi
Discussion	ccclxxi
Item	ccclxxi
Syntax	ccclxxi
Parameter	ccclxxii
Return value	ccclxxii
Discussion	ccclxxii
LicensesCollection	ccclxxii
Syntax	ccclxxii
Discussion	ccclxxii
Methods	ccclxxii
Properties	ccclxxiii
Count	ccclxxiii

Table of Contents

Syntax	ccclxxiii
Property value	ccclxxiii
Find	ccclxxiii
Syntax	ccclxxiii
Parameter	ccclxxiii
Return value	ccclxxiii
Discussion	ccclxxiii
GetEnumerator	ccclxxiv
Syntax	ccclxxiv
Return value	ccclxxiv
GetLicensedCount	ccclxxiv
Syntax	ccclxxiv
Parameter	ccclxxiv
Return value	ccclxxiv
GetUsedCount	ccclxxiv
Syntax	ccclxxiv
Parameter	ccclxxiv
Return value	ccclxxv
Exceptions	ccclxxv
HasEvaluation	ccclxxv
Syntax	ccclxxv
Property value	ccclxxv
HasMachineLicense	ccclxxv
Syntax	ccclxxv
Property value	ccclxxv
Item	ccclxxv
Syntax	ccclxxv
Parameter	ccclxxv
Return value	ccclxxvi
Map	ccclxxvi
Syntax	ccclxxvi
Discussion	ccclxxvi
Methods	ccclxxvi
Properties	ccclxxvii
Add	ccclxxvii
Syntax	ccclxxviii
Parameters	ccclxxviii
Return value	ccclxxviii
Discussion	ccclxxviii
Example	ccclxxviii
Commit	ccclxxix
Syntax	ccclxxix
Discussion	ccclxxix
Exceptions	ccclxxix

Table of Contents

Example	ccclxxix
Exists	ccclxxx
Syntax	ccclxxx
Parameter	ccclxxx
Return value	ccclxxx
Get	ccclxxx
Syntax	ccclxxx
Parameter	ccclxxx
Return value	ccclxxx
Example	ccclxxx
GetById	ccclxxxi
Syntax	ccclxxxi
Parameter	ccclxxxi
Return value	ccclxxxi
GetDirectoryEntry	ccclxxxi
Syntax	ccclxxxi
Return value	ccclxxxi
Discussion	ccclxxxi
GetEnumerator	ccclxxxii
Syntax	ccclxxxii
Return value	ccclxxxii
GetRedirectMap	ccclxxxii
Syntax	ccclxxxii
Parameter	ccclxxxii
Return value	ccclxxxii
Import	ccclxxxii
Syntax	ccclxxxii
Parameters	ccclxxxii
Return value	ccclxxxiii
Discussion	ccclxxxiii
Exceptions	ccclxxxiii
Example	ccclxxxiii
IsReadable	ccclxxxiii
Syntax	ccclxxxiii
Property value	ccclxxxiii
Discussion	ccclxxxiii
Example	ccclxxxiii
IsWritable	ccclxxxiv
Syntax	ccclxxxiv
Property value	ccclxxxiv
Discussion	ccclxxxiv
Example	ccclxxxiv
Name	ccclxxxiv
Syntax	ccclxxxv

Table of Contents

Property value	ccclxxxv
Discussion	ccclxxxv
Exceptions	ccclxxxv
Example	ccclxxxv
Remove	ccclxxxv
Syntax	ccclxxxv
Parameter	ccclxxxv
Exceptions	ccclxxxv
Example	ccclxxxvi
RemoveById	ccclxxxvi
Syntax	ccclxxxvi
Parameter	ccclxxxvi
Exceptions	ccclxxxvi
Store	ccclxxxvi
Syntax	ccclxxxvi
Property value	ccclxxxvi
Type	ccclxxxvi
Syntax	ccclxxxvii
Property value	ccclxxxvii
Discussion	ccclxxxvii
Example	ccclxxxvii
MzRoleAssignment	ccclxxxvii
Syntax	ccclxxxvii
Methods	ccclxxxvii
Properties	ccclxxxviii
GetComputer	ccclxxxix
Syntax	ccclxxxix
Return value	ccclxxxix
Exceptions	ccclxxxix
NetworkAccess	ccclxxxix
Syntax	ccclxxxix
Properties	ccclxxxix
Discussion	cccxc
Priority	cccxc
Syntax	cccxc
Property value	cccxc
Discussion	cccxc
RequirePassword	cccxc
Syntax	cccxc
Property value	cccxc
RunAs	cccxc
Syntax	cccxi
Property value	cccxi
Discussion	cccxi

Table of Contents

RunAsList	cccxcxi
Syntax	cccxcxi
Property value	cccxcxi
Discussion	cccxcxi
RunAsType	cccxcii
Syntax	cccxcii
Property value	cccxcii
Discussion	cccxcii
NetworkAccesses	cccxcii
Syntax	cccxcii
Methods	cccxcii
GetEnumerator	cccxcii
Syntax	cccxciii
Return value	cccxciii
Pam	cccxciii
Syntax	cccxciii
Discussion	cccxciii
Methods	cccxciii
Properties	cccxciii
Application	cccxciv
Syntax	cccxciv
Property value	cccxciv
Exceptions	cccxciv
Example	cccxciv
Pams	cccxcv
Syntax	cccxcv
Methods	cccxcv
GetEnumerator	cccxcv
Syntax	cccxcv
Return value	cccxcv
Right	cccxcv
Syntax	cccxcv
Methods	cccxcv
Properties	cccxcvi
Commit	cccxcvi
Syntax	cccxcvi
Discussion	cccxcvi
Exceptions	cccxcvi
Delete	cccxcvi
Syntax	cccxcvii
Exceptions	cccxcvii
Description	cccxcvii
Syntax	cccxcvii
Property value	cccxcvii

Table of Contents

IsReadable	cccxcvii
Syntax	cccxcvii
Property value	cccxcvii
IsWritable	cccxcvii
Syntax	cccxcvii
Property value	cccxcvii
Name	cccxcvii
Syntax	cccxcvii
Property value	cccxcviii
Exceptions	cccxcviii
Zone	cccxcviii
Syntax	cccxcviii
Property value	cccxcviii
Role	cccxcviii
Syntax	cccxcviii
Methods	cccxcviii
Properties	cccxcix
AddCommand	cd
Syntax	cd
Parameter	cd
Discussion	cd
Exceptions	cd
Example	cd
AddNetworkAccess	cdi
Syntax	cdi
Parameter	cdi
Discussion	cdi
Exceptions	cdi
AddPamAccess	cdi
Syntax	cdii
Parameter	cdii
Discussion	cdii
Exceptions	cdii
AddSsh	cdii
Syntax	cdii
Parameter	cdii
Discussion	cdii
Exceptions	cdii
AddWindowsApplication	cdii
Syntax	cdii
Parameter	cdiii
Discussion	cdiii
Exceptions	cdiii
AddWindowsDesktop	cdiii

Table of Contents

Syntax	cdiii
Parameter	cdiii
Discussion	cdiii
Exceptions	cdiii
Example	cdiii
AllowLocalUser	cdiv
Syntax	cdiv
Property value	cdiv
ApplicableTimeHexString	cdiv
Syntax	cdiv
Property value	cdiv
Discussion	cdiv
Exceptions	cdv
Assign	cdv
Syntax	cdv
Parameters	cdv
Return value	cdv
Discussion	cdv
Exceptions	cdv
Commit	cdvi
Syntax	cdvi
Discussion	cdvi
Exceptions	cdvi
Delete	cdvi
Syntax	cdvi
Exceptions	cdvi
Description	cdvi
Syntax	cdvi
Property value	cdvii
GetCommands	cdvii
Syntax	cdvii
Return value	cdvii
GetNetworkAccesses	cdvii
Syntax	cdvii
Return value	cdvii
GetPamAccesses	cdvii
Syntax	cdvii
Return value	cdvii
GetSshRights	cdvii
Syntax	cdvii
Return value	cdvii
GetWindowsApplications	cdviii
Syntax	cdviii
Return value	cdviii

Table of Contents

GetWindowsDesktops	cdviii
Syntax	cdviii
Return value	cdviii
Guid	cdviii
Syntax	cdviii
Property value	cdviii
IsApplicable	cdviii
Syntax	cdviii
Parameter	cdviii
Return value	cdix
Discussion	cdix
IsReadable	cdix
Syntax	cdix
Property value	cdix
Discussion	cdix
IsWritable	cdix
Syntax	cdix
Property value	cdix
Discussion	cdix
Name	cdix
Syntax	cdix
Property value	cdx
Exceptions	cdx
RemoveAllRights	cdx
Syntax	cdx
Discussion	cdx
RemoveCommand	cdx
Syntax	cdx
Parameter	cdx
Discussion	cdx
RemoveNetworkAccess	cdx
Syntax	cdx
Parameter	cdx
Discussion	cdxi
RemovePamAccess	cdxi
Syntax	cdxi
Parameter	cdxi
Discussion	cdxi
RemoveSshRight	cdxi
Syntax	cdxi
Parameter	cdxi
Discussion	cdxi
RemoveWindowsApplication	cdxi
Syntax	cdxii

Table of Contents

Parameter	cdxii
Discussion	cdxii
RemoveWindowsDesktop	cdxii
Syntax	cdxii
Parameter	cdxii
Discussion	cdxii
SetApplicableDay	cdxii
Syntax	cdxii
Parameter	cdxii
Discussion	cdxiii
SetApplicableHour	cdxiii
Syntax	cdxiii
Parameter	cdxiii
Discussion	cdxiii
SystemRights	cdxiii
Syntax	cdxiv
Property value	cdxiv
Discussion	cdxiv
Zone	cdxiv
Syntax	cdxiv
Property value	cdxiv
RoleAssignment	cdxv
Syntax	cdxv
Methods	cdxv
Properties	cdxv
Commit	cdxvi
Syntax	cdxvi
Discussion	cdxvi
Exceptions	cdxvi
Delete	cdxvi
Syntax	cdxvi
Exceptions	cdxvi
EndTime	cdxvi
Syntax	cdxvii
Property value	cdxvii
Exceptions	cdxvii
GetTrustee	cdxvii
Syntax	cdxvii
Return value	cdxvii
Exceptions	cdxvii
Id	cdxvii
Syntax	cdxvii
Property value	cdxvii
IsRoleOrphaned	cdxvii

Table of Contents

Syntax	cdxvii
Property value	cdxvii
IsTrusteeOrphaned	cdxviii
Syntax	cdxviii
Property value	cdxviii
LocalTrustee	cdxviii
Syntax	cdxviii
Property value	cdxviii
Exceptions	cdxviii
Role	cdxviii
Syntax	cdxviii
Property value	cdxviii
Exceptions	cdxix
StartTime	cdxix
Syntax	cdxix
Property value	cdxix
Exceptions	cdxix
TrusteeDn	cdxix
Syntax	cdxix
Property value	cdxix
Exceptions	cdxix
TrusteeType	cdxix
Syntax	cdxix
Property value	cdxx
Exceptions	cdxx
Validate	cdxx
Syntax	cdxx
Exceptions	cdxx
RoleAssignments	cdxxi
Syntax	cdxxi
Methods	cdxxi
GetEnumerator	cdxxi
Syntax	cdxxi
Return value	cdxxi
Exceptions	cdxxi
Roles	cdxxi
Syntax	cdxxi
Methods	cdxxi
GetEnumerator	cdxxii
Syntax	cdxxii
Return value	cdxxii
Exceptions	cdxxii
Ssh	cdxxii
Syntax	cdxxii

Table of Contents

Discussion	cdxxii
Methods	cdxxii
Properties	cdxxii
Application	cdxxiii
Syntax	cdxxiii
Property value	cdxxiii
Exceptions	cdxxiii
Sshs	cdxxiii
Syntax	cdxxiii
Methods	cdxxiii
GetEnumerator	cdxxiii
Syntax	cdxxiv
Return value	cdxxiv
Store	cdxxiv
Syntax	cdxxiv
Discussion	cdxxiv
Methods	cdxxiv
Properties	cdxxiv
Attach	cdxxv
Syntax	cdxxv
Parameters	cdxxv
Exceptions	cdxxv
Example	cdxxv
Create	cdxxvi
Syntax	cdxxvi
Parameters	cdxxvi
Return value	cdxxvi
Discussion	cdxxvi
Exceptions	cdxxvi
Example	cdxxvii
Delete	cdxxvii
Syntax	cdxxvii
Parameter	cdxxvii
Exceptions	cdxxviii
Example	cdxxviii
Exists	cdxxviii
Syntax	cdxxviii
Parameter	cdxxviii
Return value	cdxxviii
Exceptions	cdxxviii
Example	cdxxviii
GetDirectoryEntry	cdxxix
Syntax	cdxxix
Return value	cdxxix

Table of Contents

Discussion	cdxxix
IsReadable	cdxxix
Syntax	cdxxix
Property value	cdxxix
Discussion	cdxxix
Exceptions	cdxxix
Example	cdxxix
IsWritable	cdxxx
Syntax	cdxxx
Property value	cdxxx
Discussion	cdxxx
Exceptions	cdxxx
Example	cdxxx
Open	cdxxxi
Syntax	cdxxxi
Parameter	cdxxxi
Return value	cdxxxi
Exceptions	cdxxxi
Example	cdxxxi
User	cdxxxii
Syntax	cdxxxii
Discussion	cdxxxii
Methods	cdxxxii
Properties	cdxxxii
AddUnixProfile	cdxxxiii
Syntax	cdxxxiii
Parameters	cdxxxiii
Return value	cdxxxiii
Discussion	cdxxxiii
Exceptions	cdxxxiii
Example	cdxxxiv
AdsIInterface	cdxxxiv
Syntax	cdxxxv
Property value	cdxxxv
Example	cdxxxv
ADsPath	cdxxxv
Syntax	cdxxxv
Property value	cdxxxv
Discussion	cdxxxv
Example	cdxxxv
Commit	cdxxxvi
Syntax	cdxxxvi
Discussion	cdxxxvi
Exceptions	cdxxxvi

Table of Contents

Example	cdxxxvi
CommitWithoutCheck	cdxxxvi
Syntax	cdxxxvi
Discussion	cdxxxvi
Exceptions	cdxxxvii
Example	cdxxxvii
GetDirectoryEntry	cdxxxvii
Syntax	cdxxxvii
Return value	cdxxxvii
Discussion	cdxxxvii
Exceptions	cdxxxvii
GetRoleAssignmentsFromDomain	cdxxxvii
Syntax	cdxxxviii
Parameters	cdxxxviii
Return value	cdxxxviii
Discussion	cdxxxviii
Example	cdxxxviii
GetRoleAssignmentsFromForest	cdxxxviii
Syntax	cdxxxviii
Parameters	cdxxxviii
Return value	cdxxxix
Discussion	cdxxxix
Example	cdxxxix
ID	cdxxxix
Syntax	cdxxxix
Property value	cdxxxix
Example	cdxxxix
Refresh	cdxl
Syntax	cdxl
Discussion	cdxl
Exceptions	cdxl
Example	cdxl
UnixProfiles	cdxl
Syntax	cdxl
Property value	cdxl
Discussion	cdxli
Example	cdxli
UserInfo	cdxli
Syntax	cdxli
Methods	cdxli
Properties	cdxlii
CandidateDN	cdxlii
Syntax	cdxlii
Property value	cdxlii

Table of Contents

Discussion	cdxlili
Commit	cdxlili
Syntax	cdxlili
Delete	cdxlili
Syntax	cdxlili
Discussion	cdxlili
Exceptions	cdxlili
Gecos	cdxlili
Syntax	cdxlili
Property value	cdxlili
GetCandidate	cdxlili
Syntax	cdxlili
Return value	cdxlili
HomeDirectory	cdxlili
Syntax	cdxliv
Property value	cdxliv
ID	cdxliv
Syntax	cdxliv
Property value	cdxliv
Import	cdxliv
Syntax	cdxliv
Parameter	cdxliv
Discussion	cdxliv
Name	cdxliv
Syntax	cdxliv
Property value	cdxliv
PrimaryGroupID	cdxliv
Syntax	cdxliv
Property value	cdxliv
Discussion	cdxliv
SetCandidate	cdxliv
Syntax	cdxliv
Parameters	cdxliv
Discussion	cdxliv
Shell	cdxliv
Syntax	cdxliv
Property value	cdxliv
Source	cdxliv
Syntax	cdxlvi
Property value	cdxlvi
Discussion	cdxlvi
Status	cdxlvi
Syntax	cdxlvi
Property value	cdxlvi

Table of Contents

Discussion	cdxlv
StatusDescription	cdxlv
Syntax	cdxlv
Property value	cdxlvii
Discussion	cdxlvii
TimeStamp	cdxlvii
Syntax	cdxlvii
Property value	cdxlvii
Example	cdxlvii
UID	cdxlvii
Syntax	cdxlvii
Property value	cdxlvii
Discussion	cdxlviii
UpdateStatus	cdxlviii
Syntax	cdxlviii
Discussion	cdxlviii
UserInfos	cdxlviii
Syntax	cdxlviii
Methods	cdxlviii
Properties	cdxlix
Count	cdxlix
Syntax	cdxlix
Property value	cdxlix
Discussion	cdxlix
Find	cdxlix
Syntax	cdxlix
Parameter	cdxlix
Return value	cdxlix
GetEnumerator	cdxlix
Syntax	cdl
Return value	cdl
IsEmpty	cdl
Syntax	cdl
Property value	cdl
Discussion	cdl
UserUnixProfile	cdl
Syntax	cdl
Discussion	cdl
Methods	cdl
Properties	cdli
ADsPath	cdlii
Syntax	cdlii
Property value	cdlii
Example	cdlii

Table of Contents

Cims	cdlii
Syntax	cdlii
Property value	cdlii
Discussion	cdliii
Commit	cdliii
Syntax	cdliii
Discussion	cdliii
Exceptions	cdliii
Example	cdliii
Delete	cdliii
Syntax	cdliii
Discussion	cdliv
Example	cdliv
GetDirectoryEntry	cdliv
Syntax	cdliv
Return value	cdliv
Discussion	cdliv
GetPrimaryGroup	cdliv
Syntax	cdliv
Return value	cdliv
Example	cdliv
HomeDirectory	cdlv
Syntax	cdlv
Property value	cdlv
Example	cdlv
ID	cdlv
Syntax	cdlv
Property value	cdlv
Example	cdlv
IsForeign	cdlvi
Syntax	cdlvi
Property value	cdlvi
Discussion	cdlvi
Example	cdlvi
IsOrphan	cdlvi
Syntax	cdlvi
Property value	cdlvi
Discussion	cdlvii
Example	cdlvii
IsReadable	cdlvii
Syntax	cdlvii
Property value	cdlvii
Discussion	cdlvii
Example	cdlvii

Table of Contents

IsSFU	cdlviii
Syntax	cdlviii
Property value	cdlviii
IsWritable	cdlviii
Syntax	cdlviii
Property value	cdlviii
Discussion	cdlviii
Example	cdlviii
Name	cdlviii
Syntax	cdlix
Property value	cdlix
Example	cdlix
PrimaryGroup	cdlix
Syntax	cdlix
Property value	cdlix
Discussion	cdlix
Example	cdlix
ProfileState	cdlix
Syntax	cdlx
Property value	cdlx
Exceptions	cdlx
Refresh	cdlx
Syntax	cdlx
Discussion	cdlx
Example	cdlx
Shell	cdlx
Syntax	cdlx
Property value	cdlx
Example	cdlx
Type	cdlxi
Syntax	cdlxi
Property value	cdlxi
Discussion	cdlxi
Example	cdlxi
UnixEnabled	cdlxii
Syntax	cdlxii
Property value	cdlxii
Discussion	cdlxii
Exceptions	cdlxii
Example	cdlxii
User	cdlxii
Syntax	cdlxii
Property value	cdlxii
UserId	cdlxii

Table of Contents

Syntax	cdlxiii
Property value	cdlxiii
Validate	cdlxiii
Syntax	cdlxiii
Discussion	cdlxiii
Exceptions	cdlxiii
Example	cdlxiii
Zone	cdlxiii
Syntax	cdlxiii
Property value	cdlxiii
Example	cdlxiv
UserUnixProfiles	cdlxiv
Syntax	cdlxiv
Discussion	cdlxiv
Methods	cdlxiv
Properties	cdlxiv
Count	cdlxv
Syntax	cdlxv
Property value	cdlxv
Example	cdlxv
GetEnumerator	cdlxv
Syntax	cdlxv
Return value	cdlxv
IsEmpty	cdlxv
Syntax	cdlxv
Property value	cdlxv
Discussion	cdlxv
Refresh	cdlxvi
Syntax	cdlxvi
Discussion	cdlxvi
WindowsApplication	cdlxvi
Syntax	cdlxvi
Methods	cdlxvi
Properties	cdlxvi
Discussion	cdlxvii
ApplicationCriteriaList	cdlxvii
Syntax	cdlxvii
Property value	cdlxvii
Example	cdlxvii
CreateApplicationCriteria	cdlxviii
Syntax	cdlxviii
Discussion	cdlxviii
Example	cdlxviii
Priority	cdlxviii

Table of Contents

Syntax	cdlxviii
Property value	cdlxviii
Discussion	cdlxix
Example	cdlxix
RequirePassword	cdlxix
Syntax	cdlxix
Property value	cdlxix
Example	cdlxix
RunAs	cdlxix
Syntax	cdlxix
Property value	cdlxix
Discussion	cdlxix
RunAsList	cdlxx
Syntax	cdlxx
Property value	cdlxx
Discussion	cdlxx
RunAsType	cdlxx
Syntax	cdlxx
Property value	cdlxx
Discussion	cdlxxi
Example	cdlxxi
WindowsApplicationCriteria	cdlxxi
Syntax	cdlxxi
Methods	cdlxxi
Properties	cdlxxi
Discussion	cdlxxiii
Argument	cdlxxiii
Syntax	cdlxxiv
Property value	cdlxxiv
CompanyName	cdlxxiv
Syntax	cdlxxiv
Property value	cdlxxiv
Example	cdlxxiv
CompanyNameMatchOption	cdlxxiv
Syntax	cdlxxiv
Property value	cdlxxiv
Example	cdlxxv
Description	cdlxxv
Syntax	cdlxxv
Property value	cdlxxv
Example	cdlxxv
FileDescriptionMatchOption	cdlxxv
Syntax	cdlxxv
Property value	cdlxxv

Table of Contents

FileDescriptionMatchOption	cdlxxvi
Syntax	cdlxxvi
Property value	cdlxxvi
FileHash	cdlxxvi
Syntax	cdlxxvi
Property value	cdlxxvi
FileName	cdlxxvi
Syntax	cdlxxvi
Property value	cdlxxvi
FileType	cdlxxvii
Syntax	cdlxxvii
Property value	cdlxxvii
Example	cdlxxvii
FileVersion	cdlxxviii
Syntax	cdlxxviii
Property value	cdlxxviii
FileVersionMatchOption	cdlxxviii
Syntax	cdlxxviii
Property value	cdlxxviii
IsArgumentCaseSensitive	cdlxxviii
Syntax	cdlxxviii
Property value	cdlxxviii
IsArgumentExactMatch	cdlxxviii
Syntax	cdlxxix
Property value	cdlxxix
LocalOwner	cdlxxix
Syntax	cdlxxix
Property value	cdlxxix
OwnerDN	cdlxxix
Syntax	cdlxxix
Property value	cdlxxix
OwnerSid	cdlxxix
Syntax	cdlxxix
Property value	cdlxxix
OwnerType	cdlxxix
Syntax	cdlxxx
Property value	cdlxxx
Path	cdlxxx
Syntax	cdlxxx
Property value	cdlxxx
ProductName	cdlxxx
Syntax	cdlxxxi
Property value	cdlxxxi
ProductNameMatchOption	cdlxxxi

Table of Contents

Syntax	cdlxxxix
Property value	cdlxxxix
ProductVersion	cdlxxxix
Syntax	cdlxxxix
Property value	cdlxxxix
ProductVersionMatchOption	cdlxxxix
Syntax	cdlxxxix
Property value	cdlxxxix
Publisher	cdlxxxix
Syntax	cdlxxxix
Property value	cdlxxxix
PublisherMatchOption	cdlxxxix
Syntax	cdlxxxix
Property value	cdlxxxix
RequireAdministrator	cdlxxxix
Syntax	cdlxxxix
Property value	cdlxxxix
SerialNumber	cdlxxxix
Syntax	cdlxxxix
Property value	cdlxxxix
SerialNumberMatchOption	cdlxxxix
Syntax	cdlxxxix
Property value	cdlxxxix
Validate	cdlxxxix
Syntax	cdlxxxix
Discussion	cdlxxxix
Exception	cdlxxxix
WindowsApplications	cdlxxxix
Syntax	cdlxxxix
Methods	cdlxxxix
GetEnumerator	cdlxxxix
Syntax	cdlxxxix
Return value	cdlxxxix
WindowsDesktop	cdlxxxix
Syntax	cdlxxxix
Properties	cdlxxxix
Discussion	cdlxxxix
Priority	cdlxxxix
Syntax	cdlxxxix
Property value	cdlxxxix
Discussion	cdlxxxix
RequirePassword	cdlxxxix
Syntax	cdlxxxix
Property value	cdlxxxix

Table of Contents

RunAs	cdlxxxvi
Syntax	cdlxxxvi
Property value	cdlxxxvi
Discussion	cdlxxxvi
RunAsList	cdlxxxvi
Syntax	cdlxxxvi
Property value	cdlxxxvii
Discussion	cdlxxxvii
RunAsType	cdlxxxvii
Syntax	cdlxxxvii
Property value	cdlxxxvii
Discussion	cdlxxxvii
WindowsDesktops	cdlxxxviii
Syntax	cdlxxxviii
Methods	cdlxxxviii
GetEnumerator	cdlxxxviii
Syntax	cdlxxxviii
Return value	cdlxxxviii
WindowsUser	cdlxxxviii
Syntax	cdlxxxviii
Methods	cdlxxxviii
Properties	cdlxxxviii
Discussion	cdlxxxix
AddUserRoleAssignment	cdlxxxix
Syntax	cdlxxxix
Parameters	cdlxxxix
Return value	cdlxxxix
Discussion	cdlxxxix
Exceptions	cdlxxxix
GetDirectoryEntry	cdxc
Syntax	cdxc
Return value	cdxc
Discussion	cdxc
Exceptions	cdxc
GetEffectiveUserRoleAssignments	cdxc
Syntax	cdxc
Return value	cdxc
Discussion	cdxc
Name	cdxc
Syntax	cdxc
Property value	cdxci
WindowsUsers	cdxci
Syntax	cdxci
Methods	cdxci

Table of Contents

GetEnumerator	cdxcici
Syntax	cdxcici
Return value	cdxcici
Zone	cdxcici
Syntax	cdxcici
Discussion	cdxcici
Methods	cdxcici
Properties	cdxciv
AddMitUser	cdxciv
Syntax	cdxciv
Parameters	cdxciv
Return value	cdxcvi
Discussion	cdxcvi
Exceptions	cdxcvi
Example	cdxcvi
AdsIInterface	cdxcvii
Syntax	cdxcvii
Property value	cdxcvii
Discussion	cdxcvii
Example	cdxcvii
ADsPath	cdxcvii
Syntax	cdxcvii
Property value	cdxcvii
Example	cdxcvii
AgentlessAttribute	cdxcviii
Syntax	cdxcviii
Property value	cdxcviii
Discussion	cdxcviii
Exceptions	cdxcviii
Example	cdxcviii
AvailableShells	cdxcviii
Syntax	cdxcviii
Property value	cdxcviii
Discussion	cdxcix
Example	cdxcix
Cims	cdxcix
Syntax	cdxcix
Property value	cdxcix
Discussion	cdxcix
Commit	cdxcix
Syntax	cdxcix
Discussion	d
Exceptions	d
Example	d

Table of Contents

CreateImportPendingGroup	d
Syntax	d
Parameters	d
Return value	di
Discussion	di
Example	di
CreateImportPendingUser	di
Syntax	di
Parameters	di
Return value	di
Discussion	dii
Example	dii
DefaultGroup	dii
Syntax	dii
Property value	dii
Discussion	dii
Example	dii
DefaultHomeDirectory	diii
Syntax	diii
Property value	diii
Discussion	diii
Example	diii
DefaultShell	diii
Syntax	diii
Property value	diii
Discussion	div
Example	div
DefaultValueZone	div
Syntax	div
Property value	div
Discussion	div
Example	div
Delete	div
Syntax	dv
Discussion	dv
Exceptions	dv
Example	dv
Description	dv
Syntax	dv
Property value	dv
Discussion	dv
Example	dv
FullName	dvi
Syntax	dvi

Table of Contents

Property value	dvi
Discussion	dvi
Example	dvi
GetComputerByDN	dvi
Syntax	dvi
Parameter	dvi
Return value	dvii
Discussion	dvii
Example	dvii
GetComputers	dvii
Syntax	dvii
Return value	dvii
Example	dvii
GetComputersContainer	dvii
Syntax	dviii
Return value	dviii
Discussion	dviii
Exceptions	dviii
GetDirectoryEntry	dviii
Syntax	dviii
Return value	dviii
Discussion	dviii
GetDisplayName	dviii
Syntax	dviii
Return value	dviii
Discussion	dix
Example	dix
GetGroupsContainer	dix
Syntax	dix
Return value	dix
Discussion	dix
Exceptions	dix
GetGroupUnixProfile	dix
Syntax	dix
Parameter	dix
Return value	dx
Discussion	dx
Exceptions	dx
Example	dx
GetGroupUnixProfileByDN	dx
Syntax	dx
Parameter	dx
Return value	dx
Discussion	dx

Table of Contents

Exceptions	dx <i>i</i>
Example	dx <i>i</i>
GetGroupUnixProfileByName	dx <i>i</i>
Syntax	dx <i>i</i>
Parameter	dx <i>i</i>
Return value	dx <i>i</i>
Discussion	dx <i>i</i>
Exceptions	dx <i>ii</i>
Example	dx <i>ii</i>
GetGroupUnixProfiles	dx <i>ii</i>
Syntax	dx <i>ii</i>
Return value	dx <i>ii</i>
Example	dx <i>ii</i>
GetImportPendingGroup	dx <i>ii</i>
Syntax	dx <i>ii</i>
Parameter	dx <i>iii</i>
Return value	dx <i>iii</i>
Discussion	dx <i>iii</i>
Example	dx <i>iii</i>
GetImportPendingGroups	dx <i>iii</i>
Syntax	dx <i>iii</i>
Return value	dx <i>iii</i>
Example	dx <i>iii</i>
GetImportPendingUser	dx <i>iv</i>
Syntax	dx <i>iv</i>
Parameter	dx <i>iv</i>
Return value	dx <i>iv</i>
Discussion	dx <i>iv</i>
Example	dx <i>iv</i>
GetImportPendingUsers	dx <i>v</i>
Syntax	dx <i>v</i>
Return value	dx <i>v</i>
Example	dx <i>v</i>
GetLocalGroupsContainer	dx <i>v</i>
Syntax	dx <i>v</i>
Return value	dx <i>v</i>
Discussion	dx <i>v</i>
Exceptions	dx <i>vi</i>
GetLocalGroupUnixProfile	dx <i>vi</i>
Syntax	dx <i>vi</i>
Parameter	dx <i>vi</i>
Return value	dx <i>vi</i>
Exceptions	dx <i>vi</i>
GetLocalGroupUnixProfileByDN	dx <i>vi</i>

Table of Contents

Syntax	dxvi
Parameter	dxvi
Return value	dxvii
GetLocalGroupUnixProfileByGid (Int32)	dxvii
Syntax	dxvii
Parameter	dxvii
Return value	dxvii
GetLocalGroupUnixProfiles	dxvii
Syntax	dxvii
Return value	dxvii
GetLocalUsersContainer	dxvii
Syntax	dxvii
Return value	dxvii
Discussion	dxvii
Exceptions	dxviii
GetLocalUserUnixProfile	dxviii
Syntax	dxviii
Parameter	dxviii
Return value	dxviii
GetLocalUserUnixProfileByDN	dxviii
Syntax	dxviii
Parameter	dxviii
Return value	dxviii
GetLocalUserUnixProfileByUid (Int32)	dxviii
Syntax	dxix
Parameter	dxix
Return value	dxix
GetLocalUserUnixProfiles	dxix
Syntax	dxix
Return value	dxix
GetUsersContainer	dxix
Syntax	dxix
Return value	dxix
Discussion	dxix
Exceptions	dxix
GetUserUnixProfileByDN	dxx
Syntax	dxx
Parameter	dxx
Return value	dxx
Discussion	dxx
Exceptions	dxx
Example	dxx
GetUserUnixProfileByName	dxx
Syntax	dxx

Table of Contents

Parameter	dxxi
Return value	dxxi
Exceptions	dxxi
Example	dxxi
GetUserUnixProfiles	dxxi
Syntax	dxxi
Return value	dxxi
Example	dxxi
GroupAutoProvisioningEnabled	dxxii
Syntax	dxxii
Property value	dxxii
Discussion	dxxii
GroupUnixProfileExists	dxxii
Syntax	dxxii
Parameter	dxxii
Return value	dxxii
Exceptions	dxxii
Example	dxxiii
ID	dxxiii
Syntax	dxxiii
Property value	dxxiii
Discussion	dxxiii
Example	dxxiii
IsHierarchical	dxxiii
Syntax	dxxiii
Property value	dxxiii
IsReadable	dxxiv
Syntax	dxxiv
Property value	dxxiv
Discussion	dxxiv
Example	dxxiv
IsSFU	dxxiv
Syntax	dxxiv
Property value	dxxiv
Discussion	dxxiv
Example	dxxiv
IsTruncateName	dxxv
Syntax	dxxv
Property value	dxxv
Discussion	dxxv
IsWritable	dxxv
Syntax	dxxv
Property value	dxxv
Discussion	dxxv

Table of Contents

Example	dxxv
Licenses	dxxvi
Syntax	dxxvi
Property value	dxxvi
LocalGroupUnixProfileExists	dxxvi
Syntax	dxxvi
Parameter	dxxvi
Return value	dxxvi
Exceptions	dxxvi
LocalUserUnixProfileExists	dxxvi
Syntax	dxxvi
Parameter	dxxvi
Return value	dxxvii
Exceptions	dxxvii
MasterDomainController	dxxvii
Syntax	dxxvii
Property value	dxxvii
Example	dxxvii
MustMaintainADGroupMembership	dxxvii
Syntax	dxxvii
Property value	dxxvii
Discussion	dxxviii
Example	dxxviii
Name	dxxviii
Syntax	dxxviii
Property value	dxxviii
Exceptions	dxxviii
Example	dxxviii
NextAvailableGID	dxxix
Syntax	dxxix
Property value	dxxix
Discussion	dxxix
Example	dxxix
NextAvailableUID	dxxix
Syntax	dxxix
Property value	dxxix
Discussion	dxxix
Example	dxxx
NextGID	dxxx
Syntax	dxxx
Property value	dxxx
Discussion	dxxx
NextUID	dxxx
Syntax	dxxx

Table of Contents

Property value	dxxx
Discussion	dxxxi
NISDomain	dxxxi
Syntax	dxxxi
Property value	dxxxi
Discussion	dxxxi
Exceptions	dxxxi
Example	dxxxi
PrecreateComputer	dxxxi
Syntax	dxxxii
Parameters	dxxxii
Return value	dxxxii
Discussion	dxxxii
PrecreateWindowsComputer	dxxxii
Syntax	dxxxiii
Parameters	dxxxiii
Return value	dxxxiii
Refresh	dxxxiii
Syntax	dxxxiii
Discussion	dxxxiii
Exceptions	dxxxiii
Example	dxxxiii
ReservedGID	dxxxiv
Syntax	dxxxiv
Discussion	dxxxiv
Example	dxxxiv
ReservedUID	dxxxiv
Syntax	dxxxiv
Discussion	dxxxiv
Example	dxxxv
Schema	dxxxv
Syntax	dxxxv
Property value	dxxxv
Discussion	dxxxv
Exceptions	dxx xvii
Example	dxx xvii
SFUDomain	dxx xvii
Syntax	dxx xvii
Property value	dxx xvii
Example	dxx xvii
UserAutoProvisioningEnabled	dxx xviii
Syntax	dxx xviii
Property value	dxx xviii
Discussion	dxx xviii

Table of Contents

UserUnixProfileExists	dxxxviii
Syntax	dxxxviii
Parameter	dxxxviii
Return value	dxxxviii
Exceptions	dxxxviii
Example	dxxxix
Version	dxxxix
Syntax	dxxxix
Property value	dxxxix
Example	dxxxix
Reading and Setting Timebox Values	dxl
Hex String	dxl
Hour Mapping	dxl
Day Mapping	dqli

About this Guide

This guide, object reference help, and sample scripts are key components of the Software Development Kit (SDK). This guide describes the basic object model that 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 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 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 Server Suite architecture, terms, and concepts, and understand how to perform administrative tasks using software and the UNIX platforms you support. This guide assumes you are familiar with the information in the *Linux/Unix Administrator Guide*.

Using this Guide

This guide discusses the SDK COM objects and provides detailed information about how Server Suite-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.

Compatibility and Limitations

The information in this guide is intended for use with the current version of 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 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 Delinea website or Support to find out if another version of this guide is available.

About the Server Suite SDK

The Server Suite 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 Server Suite-specific data in Active Directory.

As part of the Server Suite SDK, this guide also provides detailed information about the underlying Server Suite data storage model and how attributes managed by Server Suite are stored in Active Directory. This information is critical for developing programs that access or modify Server Suite 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 Delinea Agent for *NIX 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 [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 Server Suite 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.

Windows Developer Tools

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

- A library of commands for access control and privilege management that run in Windows PowerShell.
- Sample scripts that use the Delinea PowerShell cmdlets to illustrate common administrative tasks.

About the Server Suite SDK

- Dynamic link libraries that expose interfaces for working with Delinea 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 [Scripting Access Control and Privilege Management with PowerShell](#).

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

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

UNIX Developer Tools

If you plan to develop programs that run on UNIX computers, the Server Suite 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 [Adedit Command Reference and Scripting Guide](#). For more information about developing scripts that use LDAP commands, see "Data Storage for Zones" on page xcvi.

Downloading the 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 Server Suite SDK as a separate software package from the Delinea Download Center under SDKs. 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 Delinea sales representative to access the SDK.

Installing the Standalone SDK Package

You can download the SDK package [separately](#) or the Windows installer package includes the SDK package that you can install. The installer is in the zip file, in the `DirectManage\SDK\` folder.

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. In the extracted files of the Windows installer package, navigate to the `DirectManage\SDK\` folder.
2. Double-click the downloaded executable to start the SDK setup program.
For example, for the 64-bit version of the 6.2.0 SDK, double click the `DirectControl_SDK-6.2.0-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 Windows API Object Model

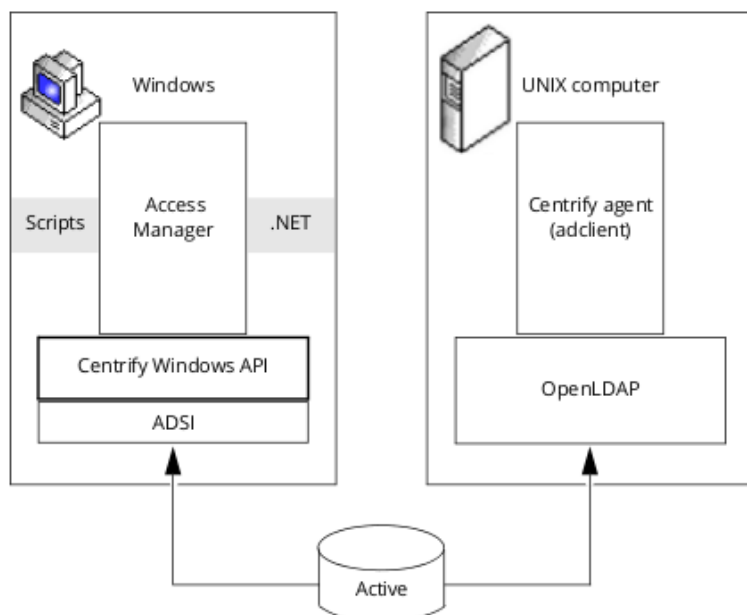
This section provides an overview of the architecture and capabilities of the object library exposed in the Server Suite Windows API included in the SDK and how you can use the objects in applications to access and manage Server Suite data on Windows computers. It includes a discussion of the Server Suite 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 Windows API Relies on COM Interfaces

On Windows computers, the Server Suite 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 Server Suite 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 Server Suite 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 Server Suite Windows API is to provide a higher level of abstraction for performing Server Suite-specific tasks than would be available if you were to call ADSI functions directly.

The following figure illustrates how the Server Suite 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 Server Suite data to the Active Directory database, how that data is stored depends on the Active Directory schema

you have installed. The Server Suite 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 Server Suite 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 Server Suite Windows API objects documented in this guide to manipulate the data.

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



Note: You can use ADSI directly instead of using the Server Suite 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 Server Suite Zones.

Administrative Tasks

Using the Server Suite 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 Server Suite objects, see the sample scripts and help included with the software package.

Although you can use the Server Suite 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 Server Suite Windows API. To learn how to create user or group objects programmatically or perform other tasks that are not provided by the Server Suite Windows API, such as changing access rights, refer to Microsoft's ADSI documentation.

Server Suite-Specific Object Classes

The Server Suite Windows API consists of several common, interdependent classes that correspond with the core elements of Server Suite-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 Server Suite Windows API.

Class	Description
AzRoleAssignment	Represents a computer-role assignment.
Cims	Initiates interaction with Active Directory. This top-level class connects to Active Directory and prepares the Active Directory domain and forest for working with Server Suite objects.
Command : Right	Represents the right to run a command, including which users and groups have that right.
Commands	Represents a collection of command rights.
Computer	Manages an individual computer account object.
ComputerGroupUnixProfiles : GroupUnixProfiles	Represents the groups in a computer zone.
ComputerRole	Manages a computer role.
ComputerRoles	Represents a collection of computer roles.
Computers	Represents a collection of computers in a zone.
ComputerUserUnixProfiles: UserUnixProfiles	Represents the users in a computer zone.
Group	Manages an individual group account object.
GroupUnixProfile	Manages the properties in the UNIX profile of a group.
GroupUnixProfiles	Represents a collection of UNIX groups in a zone.
HierarchicalGroup : GroupUnixProfile	Manages the properties in the UNIX profile of a group in a hierarchical zone.
HierarchicalUser : UserUnixProfile	Manages the properties in the UNIX profile of a user in a hierarchical zone.
HierarchicalZone : Zone	Represents a hierarchical zone.
HierarchicalZoneComputer: Computer	Manages the properties in the profile of a computer object joined to a hierarchical zone.
HierarchicalZone : RoleAssignment	Manages a zone-level role assignment in a hierarchical zone.
InheritedRoleAsg	Represents an inherited role assignment.

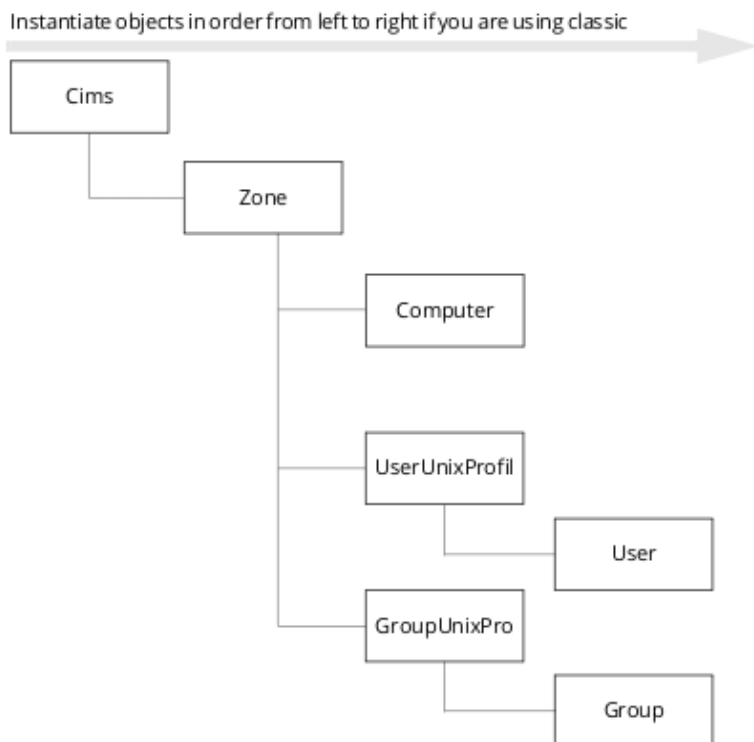
Class	Description
Key	Represents a license key.
Keys	Represents a collection of Server Suite license keys.
License	Represents a Server Suite license.
Licenses	Represents a collection of Server Suite licenses in a license container object.
LicensesCollection	Manages all the Server Suite licenses in all of the Licenses parent containers defined for a forest.
MzRoleAssignment : RoleAssignment	Represents a computer-level role assignment.
NetworkAccesses	Represents a collection of network access rights.
Pam : Right	Represents a PAM (Pluggable Authentication Module) 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 Server Suite 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.

Class	Description
WindowsUser	Represents a Windows user.
WindowsUsers	Represents a collection of Windows users.
Zone	Represents a Server Suite zone, including the users, groups, and computers that have been added to the zone.

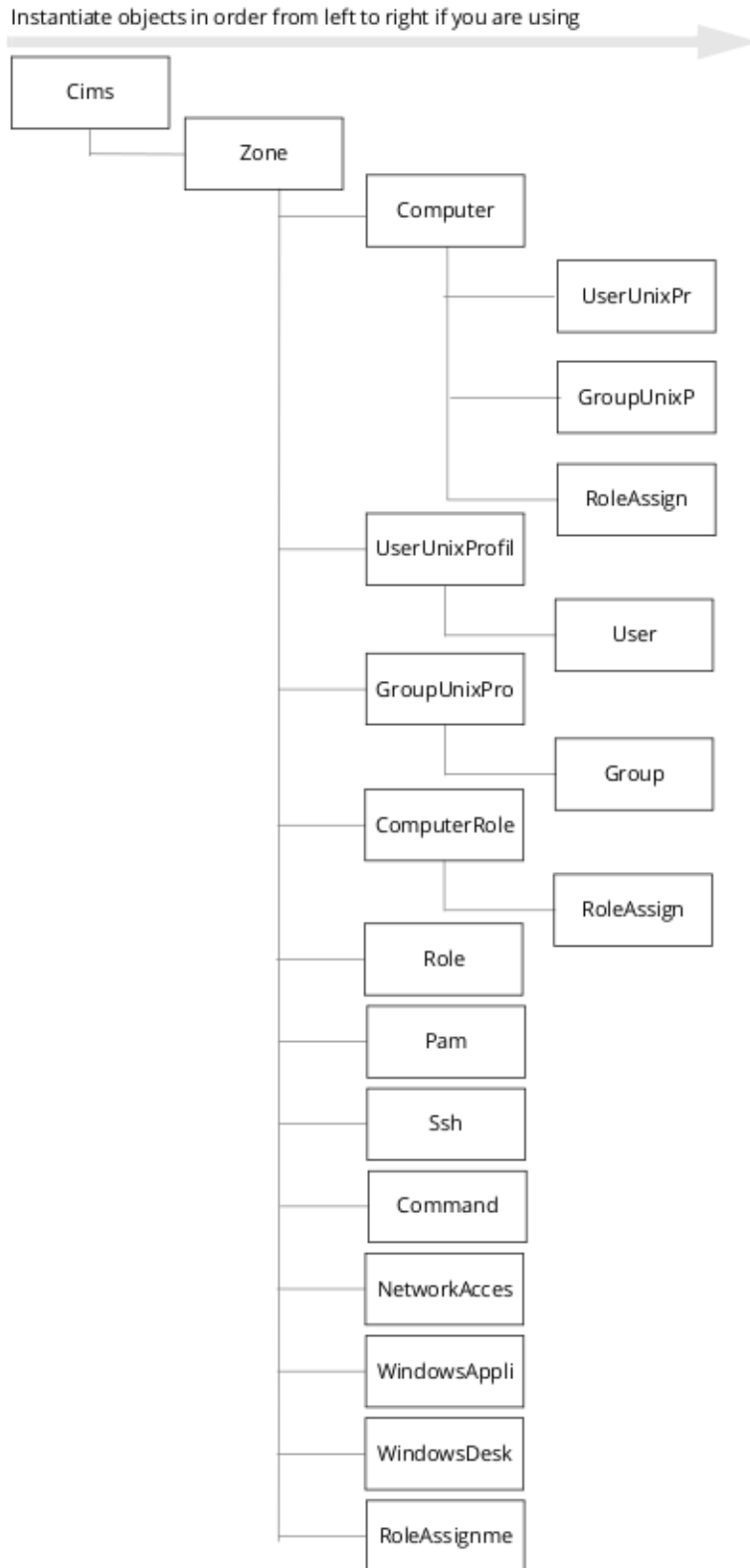
In addition to these objects, there are optional objects for managing and manipulating NIS maps and NIS map entries in Active Directory. For an overview of those objects, see [Working With NIS Maps](#). For more information about all of the objects that enable you to manipulate Server Suite-specific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

Creating Objects in the Proper Order

Server Suite 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 Server Suite-related objects:



Overview of the Windows API Object Model



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 Server Suite 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, Server Suite-specific objects are implemented as IADs interfaces. Using interfaces for the Server Suite 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 Server Suite-specific objects that are implemented as interface objects have the same names as the classes in Server Suite-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 Server Suite 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 Server Suite 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 Server Suite 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 Server Suite 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 Server Suite 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 Server Suite 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 Server Suite 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 Server Suite Network Information Service (adnisd) to receive the data.

For more detailed information about working with NIS maps and the Server Suite Network Information Service, see [Planning and Deployment Guide](#) and [Network Information Services Administration](#).

The Server Suite 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 API Calls

To handle Server Suite tasks programmatically, you can write programs that call Server Suite 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 Server Suite 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 Server SuiteWindows API.

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

Overview of the Windows API Object Model

1. Verify that the computer you are using has Access Manager console or the Server Suite Windows API Runtime environment from the Server Suite 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

Server Suite 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 Server Suite 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 Server SuiteWindows API Runtime environment from the Server Suite 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,c.
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:

```
$usrname = "administrator";
$passwd = "1234abcepassword";
```

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 c Windows API, see Scripting Access Control and Privilege Management with PowerShell.

Using Visual Studio C#

The following steps describe how to call the Server Suite 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 Server SuiteWindows API Runtime environment from the Server Suite 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

    if (zone.IsHierarchical &&
        !
        ((Centrify.DirectControl.API.CDC50.UserUnixProfile)user).IsUidDefined)
    {
        uid = "<Empty>";
    }
    else
```

```
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.

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 Server Suite 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.

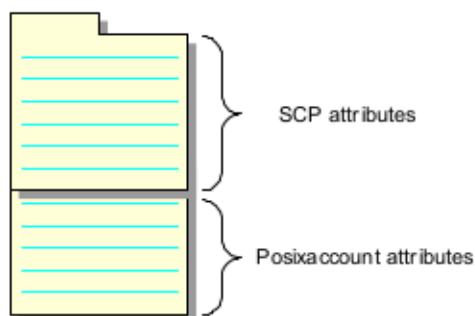
5. Get the `User` object by passing the `IADsUser` object you obtained in the previous step to `cims.GetUser(x)`.
6. With the `User` object, you can use `User.AddUnixProfile()` to add the zone profile.

Data Storage for Zones

This section provides a detailed description of how Server Suite-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.

Classic RFC 2307 Zones (3.x, 4.x)

The classic RFC 2307-compatible zone is similar to the classic Server Suite 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, Server Suite 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 Server Suite 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 Server Suite 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.

Data Storage for Zones

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 Server Suite zones](#).

Zone attribute	Stored in Active Directory attribute
ZoneName	cn:ZoneName For example: cn:default
ZoneVersion	displayName:Zoneversion The valid values are: zoneversion \(\$CimsZoneVersion3 for RFC 2307 zones, compatible with Server Suite, version 3.x \(\$CimsZoneVersion4 for standard classic zones. \(\$CimsZoneVersion5 for classic RFC 2307 zones, compatible with Server Suite, version 4.x For example: displayName:\\$CimsZoneVersion5
Description	description:value For example: description:description:Pilot EMEA
NextUid	description:uidnext:value For example: description:uidnext:12098
NextGid	description:gidnext:value For example: description:gidnext:12098
ReservedUids	description:uidreserved:value For example: description:uidreserved:0-99:501
ReservedGids	description:gidreserved:value For example: description:gidreserved:1000-2500
Availableshells	description:availableshells:value For example: description:availableshells:/bin/sh
DefaultHomeDirectory	description:defaultthome:value For example: description:defaultthome:/nfs/\\${user}
DefaultShell	description:defaultshell:value For example: description:defaultshell:/bin/bash
DefaultGroup	description:defaultgid:value For example: description:defaultgid:12098
ZoneType	schema:Dynamic_Schema_Version The valid values for classic RFC 2307compliant zones are: CDC_RFC_2307 (3.x compatible). CDC_RFC_2307_2 (4.x compatible). 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 subcontainer of the zone: the `serviceConnectionPoint` object class and the `posixAccount` object class.

User attribute	Stored in Active Directory attribute
UnixName	cn:userlogin and uid:userlogin For example: uid:cain
UserVersion	displayName:UserVersion This attribute determines compatibility between a user profile object and the Access Manager console. The only valid value for this attribute is \ \$CimsUserVersion3. For example: displayName:\ \$CimsUserVersion3
Uid	uidNumber:value For example: uidNumber:458
Gid	gidNumber:value For example: gidNumber:458
Home	unixHomeDirectory:value For example: unixHomeDirectory:/home/shear
Shell	loginShell:value For example: loginShell:/bin/bash
ParentLink	managedBy:DN_ActiveDirectoryUser 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: managedBy:cn=ben'lau,cn=users,dc=ice,dc=net If the zone does not need to be compatible with older versions of Server Suite software, you can use the keywords attribute and parentLink pseudo-attribute to specify the security identifier (SID) of the parent Active Directory user object. For example: keywords:parentLink:S-n-n-nn-nnn..
UnixEnabled	keywords:unix_enabled:value For example: keywords:unix_enabled:True
ForeignForest	keywords:foreign:value This attribute indicates whether a user in a zone is from an external forest. For example: keywords:foreign:False



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
UnixName	cn:GroupName For example: cn:performx
GroupVersion	displayName:GroupVersion This attribute determines compatibility between a group profile object and the Access manager console. The only valid value for this attribute is \ \$CimsGroupVersion3. For example: displayName:\ \$CimsGroupVersion3
Gid	gidNumber:value For example: gidNumber:458

Group attribute	Stored in Active Directory attribute
ParentLink	managedBy:DN_ActiveDirectoryGroup 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 Server Suite software, you can use the keywords attribute and parentLink pseudo-attribute to specify the security identifier (SID) of the parent Active Directory group object. For example: keywords:parentLink:S-n-n-nn-nnn..
UnixEnabled	keywords:unix_enabled:value For example: keywords:unix_enabled:True
ForeignForest	keywords:foreign:value This attribute indicates whether a group in a zone is from an external forest. For example: keywords:foreign:False



Note: The posixGroup group membership attributes are not set. Server Suite 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 pseudoattributes for this object are stored in the keywords attribute.

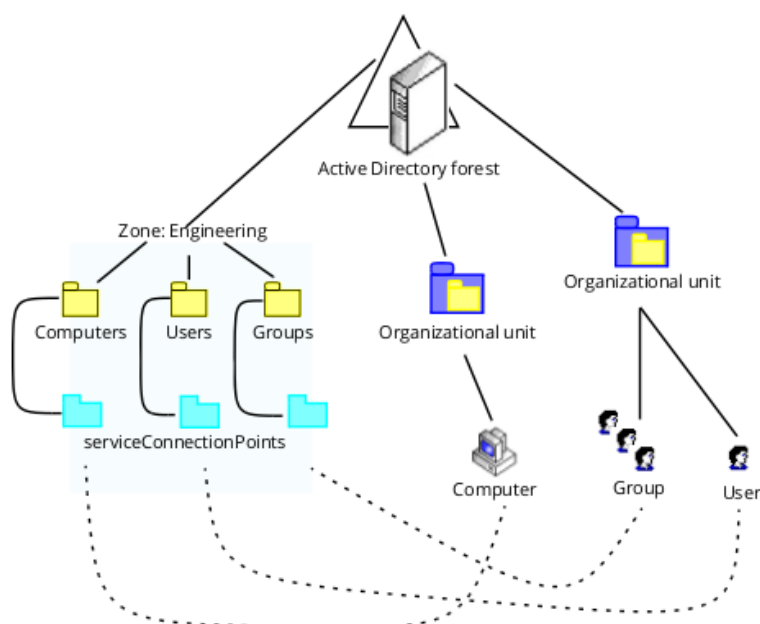
Computer attribute	Stored in Active Directory attribute
UnixName	name:ComputerName Normally, computer attribute values are set by the adjoin process and do not need to be modified. For example: name:magnolia.ajax.org
ComputerVersion	displayName:ComputerVersion This attribute determines compatibility between a computer profile object and the Access Manager console. The only valid value for this attribute is \$CimsComputerVersion3. For example: displayName:\$CimsComputerVersion3
ParentLink	managedBy:DN_ActiveDirectoryGroup 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. For example: managedBy:cn=hr,cn=computers,dc=ajax,dc=org If the zone does not need to be compatible with older versions of Server Suite software, you can use the keywords attribute and parentLink pseudo-attribute to specify the security identifier (SID) of the parent Active Directory computer object. For example: keywords:parentLink:S-n-n-nn-nnn..
AgentVersion	keywords:agentVersion:value For example: keywords:agentVersion:CentrifyDC 4.1 > Note: This attribute is set as a keywords attribute if you are using Server Suite, versions 3.0.x through 4.1.x. With the Server Suite Agent, version 4.2 or later, the AgentVersion is stored using the operatingSystemServicePack attribute of the computer object.

Computer attribute	Stored in Active Directory attribute
CpuCount	keywords:cpus:valueFor example: keywords:cpus:2 > Note: This attribute is only set if you are using Server Suite, versions 3.0.x through 4.1.x. The attribute is not set or updated for computers with the Server Suite Agent, version 4.2 or later.
JbossEnabled	keywords:jboss_enabled:value This attribute indicates whether the computer hosts JBoss applications. For example: keywords:jboss_enabled:False
TomcatEnabled	keywords:tomcat_enabled:value This attribute indicates whether the computer hosts Tomcat applications. For example: keywords:tomcat_enabled:False
UnixEnabled	keywords:unix_enabled:value For example: keywords:unix_enabled:True
webLogicEnabled	keywords:weblogic_enabled:value This attribute indicates whether the computer hosts WebLogic applications. For example: keywords:WEBLOGIC_enabled:True
webSphereEnabled	keywords:websphere_enabled:value This attribute indicates whether the computer hosts WebSphere applications. For example: keywords:websphere_enabled:True

Classic Zones (2.x, 3.x, 4.x)


In classic Server Suite 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.



Data Storage for Zones

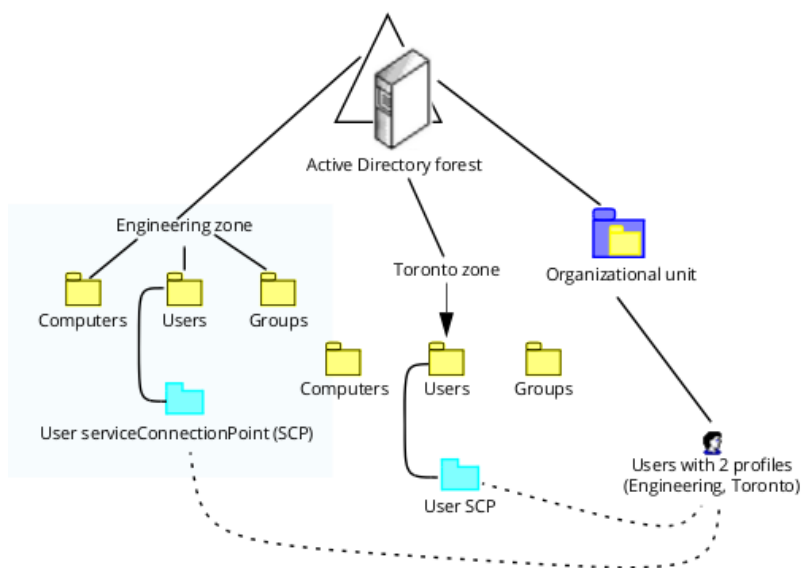
Within each of the sub-containers, there are `serviceConnectionPoint` (SCP) objects. The `serviceConnectionPoint` (SCP) objects contain the Server Suite attributes for each user, group, or computer defined for the zone. Each of user, group, or computer `serviceConnectionPoint` objects also has a link 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 Server Suite 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:

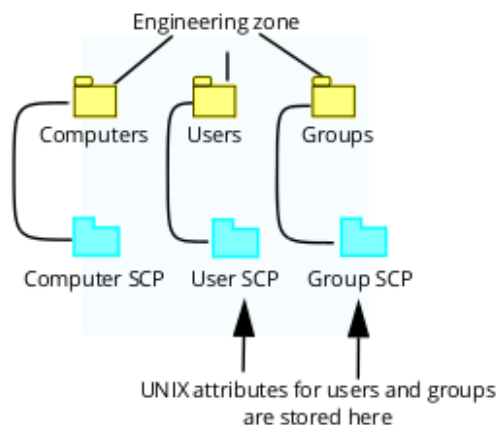
- It enables a single Active Directory user to have many different UNIX profiles.
- It enables you to delegate administrative tasks to users and groups on a zone-by-zone basis.

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 Server Suite 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.

Data Storage for Zones



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 Server Suite, 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 Server Suite, version 2.x, but discontinued because there are cases when it is not possible to set the `managedBy` attribute.

Zone Attributes in Classic 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 pseudoattributes using the Active Directory description attribute. The following table summarizes how zone attributes are stored in Active Directory for Server Suite zones.

Zone attribute	Stored in Active Directory attribute
ZoneName	cn:ZoneName For example: cn:default
ZoneVersion	displayName:Zoneversion This attribute determines compatibility between a zone object and the Access Manager console. The valid values are: \\$CimsZoneVersion2 for zones compatible with Server Suite versions 2.x and 3.x \\$CimsZoneVersion3 for RFC 2307 classic zones compatible with Server Suite versions 2.x and 3.x \\$CimsZoneVersion4 for classic zones compatible with Server Suite, version 4.x \\$CimsZoneVersion5 for hierarchical zones For example: displayName: \\$CimsZoneVersion5
Description	description:description:value For example: description:description:Pilot EMEA
NextUid	description:uidnext:value For example: description:uidnext:12098

Zone attribute	Stored in Active Directory attribute
NextGid	description:gidnext:value For example: description:gidnext:12098
ReservedUids	description:uidreserved:value 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 (-). For example: description:uidreserved:0-99:501
ReservedGids	description:gidreserved:value This attribute has the same format as the reserveduids attribute. For example: description:gidreserved:1000-2500
Availableshells	description:availableshells:value This attribute can be a multi-valued list of shell names, using a colon as the separator. For example: description:availableshells:/bin/sh
DefaultHomeDirectory	description:defaulthome:value For example: description:defaulthome:/nfs/\\\${user}
DefaultShell	description:defaultshell:value For example: description:defaultshell:/bin/bash
DefaultGroup	description:defaultgid:value For example: description:defaultgid:12098
ZoneType	schema:Dynamic_Schema_Version This attribute identifies the schema layout a zone object uses. The valid values are: Dynamic_Schema_1_0 for Server Suite, version 1.0, zones. This schema type is obsolete for version 2.x and later. Dynamic_Schema_2_0 for classic Delinea zones, 2.x and 3.x compatible. Dynamic_Schema_3_0 for classic Delinea zones, 3.x and 4.x compatible. Dynamic_Schema_5_0 for hierarchical Delinea zones, 5.x compatible. SFU_3_0 for SFU zones with the Microsoft Services for UNIX (SFU), version 3.x, schema extension. SFU_4_0 for SFU zones with the Microsoft Services for UNIX (SFU), version 4.x, schema extension. CDC_RFC_2307 for classic RFC 2307 compliant zones, Server Suite 2.x and 3.x compatible. CDC_RFC_2307_2 for classic RFC 2307 compliant zones, Server Suite 4.x compatible. schema:Dynamic_Schema_Version0 for hierarchical RFC 2307 compliant zones, Server Suite 5.x compatible. For example: schema:Dynamic_Schema_Version1

User Attributes in Classic Zones

A user extension object is a serviceConnectionPoint object that is created in the Users sub-container of the zone. The pseudoattributes for this object are stored in the keywords attribute.

User attribute	Stored in Active Directory attribute
UnixName	cn:userlogin For SCP objects, the Name 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. For example: cn:cain
UserVersion	displayName:UserVersion This attribute determines compatibility between a user profile object and the Access Manager console. The only valid value for this attribute is \ \$CimsUserVersion2. For example: displayName:\ \$CimsUserVersion2
ParentLink	managedBy:DN_ActiveDirectoryUser You can use the managedBy or keywords attribute to store the 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 user object. For example: managedBy:cn=ben.lau,cn=users,dc=ice,dc=net If the zone does not need to be compatible with older versions of Server Suite software, you can use the keywords attribute and parentLink pseudo-attribute to specify the security identifier (SID) of the parent Active Directory user object. For example: keywords:parentLink:S-n-n-nn-nnn..
Uid	keywords:uid:value For example: keywords:uid:458
Gid	keywords:gid:value For example: keywords:gid:458
Home	keywords:home:value For example: keywords:home:/home/shear
Shell	keywords:shell:value For example: keywords:shell:/bin/bash
UnixEnabled	keywords:unix_enabled:value For example: keywords:unix_enabled:False
ForeignForest	keywords:foreign:value 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 Zones

A group extension object is a serviceConnectionPoint object that is created in the Groups sub-container of the zone. The pseudoattributes for this object are stored in the keywords attribute.

Group attribute	Stored in Active Directory attribute
UnixName	name:GroupName For SCP objects, the Name attribute is the same as the CN attribute. Either attribute can be set, but attribute use should be consistent with other objects. For example: name:performx
GroupVersion	displayName:GroupVersion This attribute determines compatibility between a group profile object and the Access manager console. The only valid value for this attribute is \ \$CimsGroupVersion3. For example: displayName:\ \$CimsGroupVersion3

Group attribute	Stored in Active Directory attribute
ParentLink	managedBy:DN_ActiveDirectoryGroup 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 Server Suite software, you can use the keywords attribute and parentLink pseudoattribute to specify the security identifier (SID) of the parent Active Directory group object. For example: keywords:parentLink:S-n-n-nn-nnn..
Gid	gid:value For example: keywords:gid:458
UnixEnabled	This attribute is only applicable in classic 4.x zones.
ForeignForest	Not supported in 3.x or 4.x.

Computer Attributes in Classic Zones

A computer extension object is a serviceConnectionPoint object that is created in the Computers sub-container of the zone. The pseudoattributes for this object are stored in the keywords attribute.

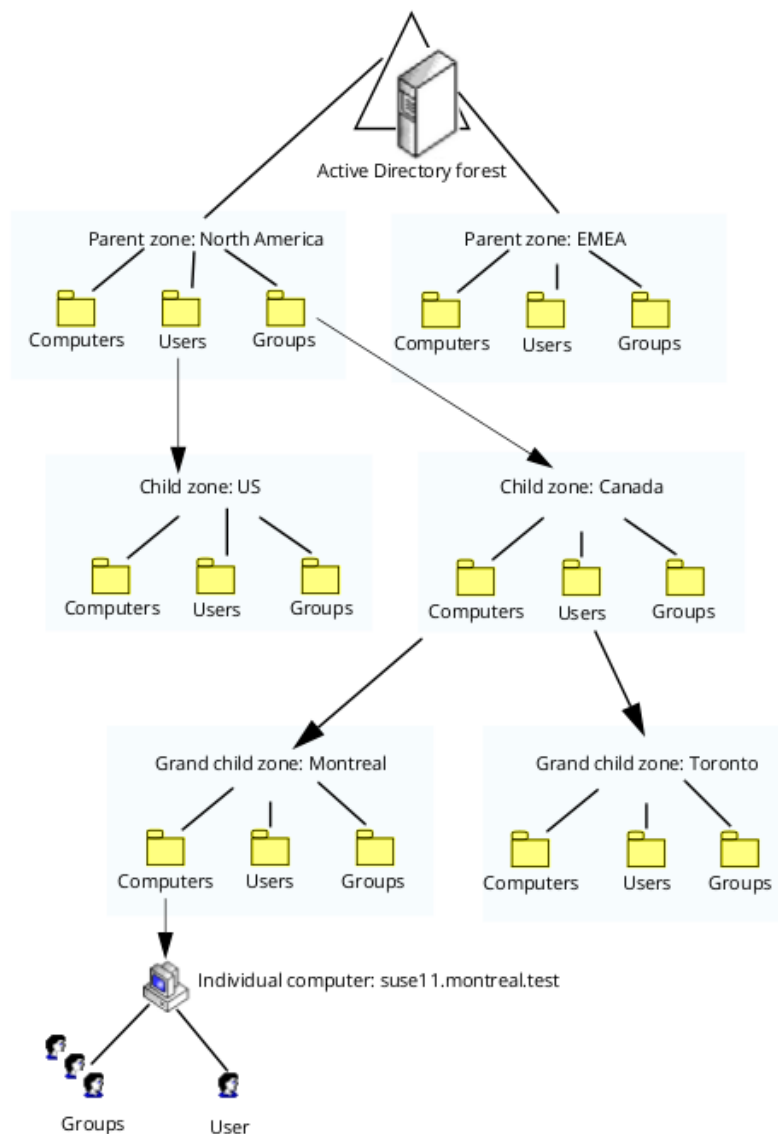
Computer attribute	Stored in Active Directory attribute
UnixName	name:ComputerName For SCP objects, the Name attribute is the same as the CN attribute. Either attribute can be set, but attribute use should be consistent with other objects. Normally, computer attribute values are set by the adjoin process and do not need to be modified. For example: name:magnolia.ajax.org
ComputerVersion	displayName:ComputerVersion This attribute determines compatibility between a computer profile object and the Access Manager console. The only valid value for this attribute is \ \$CimsComputerVersion3. For example: displayName:\ \$CimsComputerVersion3
ParentLink	managedBy:DN_ActiveDirectoryGroup 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. For example: managedBy: cn=hr,cn=computers,dc=ajax,dc=org If the zone does not need to be compatible with older versions of Server Suite software, you can use the keywords attribute and parentLink pseudo-attribute to specify the security identifier (SID) of the parent Active Directory computer object. For example: keywords:parentLink:S-n-n-nn-nnn
AgentVersion	keywords:agentVersion:value For example: keywords:agentVersion:CentrifyDC 4.1 Note This attribute is set as a keywords attribute if you are using Delinea, versions 3.0.x through 4.1.x. With the Delinea Agent, version 4.2 or later, the agentVersion is stored using the operatingSystemServicePack attribute of the computer object.

Computer attribute	Stored in Active Directory attribute
CpuCount	keywords:cpus:value For example: keywords:cpus:2 Note This attribute is only set if you are using Server Suite, versions 3.0.x through 4.1.x. The attribute is not set or updated for computers with the Server Suite Agent, version 4.2 or later.
JbossEnabled	keywords:jboss_enabled:value This attribute indicates whether the computer hosts JBoss applications. For example: keywords:jboss_enabled:False
TomcatEnabled	keywords:tomcat_enabled:value This attribute indicates whether the computer hosts Tomcat applications. For example: keywords:tomcat_enabled:False
UnixEnabled	keywords:unix_enabled:value For example: keywords:unix_enabled:True
webLogicEnabled	keywords:weblogic_enabled:value This attribute indicates whether the computer hosts WebLogic applications. For example: keywords:jboss_enabled:True
webSphereEnabled	keywords:websphere_enabled:value This attribute indicates whether the computer hosts WebSphere applications. For example: keywords:websphere_enabled:True

Hierarchical Zones (5.x)

In hierarchical Server Suite 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 Server Suite 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 Server Suite 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 Zones (2.x, 3.x, 4.x)" on page ci for a general description of the way Server Suite 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 pseudoattributes using the Active Directory description attribute. The following table summarizes how zone attributes are stored in Active Directory for hierarchical Delinea zones.

Zone attribute	Stored in Active Directory attribute	Inherited
ZoneName	cn:ZoneName For example: cn:global	No
Description	description:description:value For example: description:description:Pilot-NA	No
AvailableShells	description:availableshells:shell1:shell2 For example: description:availableshells:/bin/sh	Yes
DefaultShell	description:defaultshell:valueor description:defaultshell:%{shell} For example: description:defaultshell:/bin/bash	Yes
DefaultHomeDirectory	description:defaulthome:value or description:defaulthome:%{home}/%{user} For example: description:defaulthome:/nfs/jsmith	Yes
UserDefaultGecos	description:defaultgecos:\\${u:cn} For example: description:defaulttgecos:\\${u:upn}	Yes
customVariable	description:%variablename:value One for each variable. For example: description:%admin:SAMAccountName	Yes
ReservedUids	description:uidreserved:value 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 (). For example: description:uidreserved:0-99:501	Yes
ReservedGids	description:gidreserved:value This attribute has the same format as the reserveduids attribute. For example: description:gidreserved:1000-2500	Yes
UserDefaultuid	description:defaultuid:value Set value to \\${uidnext} to use the zone's cram attribute uidnext. The cram attribute is where the key-value pairs ("name:value") are stored. Set value to \\${autosid} to generate the UID from the domain SID and user RID. For example: description:defaultuid:\\${autosid}	Yes
DefaultGroup	description:defaultgid:value Set value to -1 to use private groups. For example: description:defaultgid:12098	Yes
UserDefaultName	description:username:\\${u:SAMAccountName}	Yes
UserDefaultRole	description:defaultrole:role-name	Yes

Zone attribute	Stored in Active Directory attribute	Inherited
GroupDefaultGid	description:defaultgroupgid:value Set value to <code>\\${gidnext}</code> to use the zone's <code>gidnext</code> attribute in classic zones. Set value to <code>\\${autosid}</code> to generate the GID from the domain SID and group RID in hierarchical zones. For example: description:defaultgid:\\${autosid}	Yes
GroupDefaultName	description:groupname:\\${g:CN}	Yes
NISDomain	description:nisdomain:name	Yes
Schema	description:schema:name Possible values are: <code>CDC_RFC_2307</code> (for a classic RFC 2307 zone) <code>CDC_GENERIC</code> (for a classic Delinea zone) <code>SFU_3_0</code> (For a classic SFU-compliant R2 schema zone) <code>SFU_3_0v1</code> (For a classic SFU-compliant zone) For example: description:Cschema:DC_GENERIC	No
AgentlessAttribute	description:pwsync:attributeName For example: description:pwsync:msSFU30Password	Yes
Licenses	description:license:guid	Yes
SFUDomain	description:alternateDomain:domain.name 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.	Yes
Parent	description:parentLink:MS-GUID@DOMAIN.NAME For example: samAccountName@domain.name[:N]: "joe@ajax.com"	No
objectType	displayName=\\$CimsZoneVersionnumber where the zone version number can be: <code>\\$CimsUserVersion4</code> for a Delinea zone <code>\\$CimsUserVersion5</code> for a RFC 2307 zone	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 pseudoattributes 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	No
Name	keywords:login:name For example: keywords:login:cain	Yes

Data Storage for Zones

uid	keywords:uid:value For example: keywords:uid:458	Yes
gid	keywords:gid:value For example: keywords:gid:458	Yes
Home	keywords:home:value For example: keywords:home:/home/shear	Yes
Shell	keywords:shell:value For example: keywords:shell:/bin/bash	Yes
Gecos	gecos:value For example: geccos:%{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 namevalue 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 true or false.
aix.admgroups	Lists the groups that the user administrates as a comma-separated list of group names.
aix.daemon	Specifies whether the user can execute programs using the the cron daemon or the system resource controller (src).
aix.rlogin	Specifies whether the user account can be logged into remotely using telnet or rlogin.
aix.su	Indicates whether other users can switch to the user account with the su command.
aix.sugroups	Lists the groups can switch to the user account as a comma-separated list of group names.
aix.tpath	Indicates the user's trusted path status.
aix.ttys	Lists the terminals that can access the account as a comma-separated list of full path names, or using ALL to indicate all terminals.

Data Storage for Zones

<code>aix.fsize</code>	Sets the soft limit for the largest file a user's process can create or extend or a value of -1 to specify unlimited 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 -1 to specify unlimited 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 -1 to specify unlimited for this attribute.
<code>aix.data</code>	Sets the soft limit for the size of a user's data segment or a value of -1 to specify unlimited 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 1 to specify unlimitedfor 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 1 to specify unlimited 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 1 to specify unlimited for this attribute.
<code>aix.umask</code>	Determines file permissions for the user using a three-digit octal value such as 022.

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 pseudoattributes for this object are stored in the keywords attribute.

Group attribute	Stored in Active Directory attribute	Inherited
<code>version</code>	<code>displayName=\\$CimsZoneVersion4</code> or, for RFC 2307 objects, use version 5: <code>displayName=\\$CimsZoneversion5</code>	No
<code>name</code>	<code>keywords:login:Name</code> For example: <code>keywords:login:ibmdba</code>	Yes
<code>gid</code>	<code>keywords:gid:value</code> For example: <code>keywords:gid:458</code> If the group is in a standard zone, the GID is stored as <code>gid:xxx</code> in the keywords attribute. If the group is in an RFC 2307 zone, the GID is stored in the schema's <code>gid</code> attribute.	Yes
<code>parentLink</code>	<code>keywords:parentLink:MS-SID</code> For example: <code>keywords:parentLink:S-1-5-21-387451290</code>	No
<code>IsMembershipRequired</code>	<code>keywords:required:value</code> For example: <code>keywords:required:true</code>	Yes
<code>InheritFromParent</code>	<code>keywords:inherit:value</code> For example: <code>keywords:inherit:true</code>	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 pseudoattributes for this object are stored in the keywords attribute.

Computer attribute	Stored in Active Directory attribute
<code>unixName</code>	<code>cn:name:ComputerName</code> Normally, the computer attribute values are set by the <code>adjoin</code> process and do not need to be modified.
<code>schemaVersion</code>	<code>displayName:Computerversion</code> 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>agentVersion</code>	<code>keywords:agentVersion:version</code> For example: <code>keywords:agentVersion:CentrifyDC 5.4.0-197</code>
<code>parentLink</code>	<code>keywords:parentLink:MS-SID</code> 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 computerspecific 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
<code>cn</code>	<code>SAMAccountName@domain.name[:N]</code>	No
<code>version</code>	<code>displayName=\\$CimsUserVersion5</code>	No
<code>Name</code>	<code>uid</code>	Yes
<code>uid</code>	<code>uidNumber</code>	Yes
<code>Gid</code>	<code>gidNumber</code>	Yes
<code>Home</code>	<code>unixHomeDirectory</code>	Yes
<code>Shell</code>	<code>loginShell</code>	Yes
<code>Gecos</code>	<code>gecos</code>	Yes

Data Storage for Zones

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 `gecos` 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](#).

The Logical Data Model for Objects

This section describes the logical data model associated with each type of Server Suite 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 Server Suite 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 Server Suite 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

Server Suite uses existing Active Directory attributes to store data. For example, most Server Suite 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 Server Suite zone, you would see results similar to the following:

```
keywords: foreign:False
keywords: gid:800keywords: 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 Server Suite 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 Server Suite zone objects.

Logical attribute	Type	Description
<code>NextUid</code>	int	The value of the next <code>UID</code> that will be allocated automatically.
<code>NextGid</code>	int	The value of the next <code>GID</code> that will be allocated automatically.
<code>ReservedUids</code>	string	<code>UIDs</code> that should not be used in the automatic allocation sequence.
<code>ReservedGids</code>	string	<code>GIDs</code> that should not be used in the automatic allocation sequence.
<code>AvailableShells</code>	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.
<code>DefaultHomeDirectory</code>	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> .
<code>DefaultShell</code>	string	The default shell to use for the zone.
<code>ZoneName</code>	string	The name of the zone.
<code>DefaultGid</code>	int	The value of the <code>GID</code> for the group profile associated with the Active Directory group used as the default group for new users.

Logical attribute	Type	Description
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 Description attribute is Business development zone, 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 Server Suite.

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. 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. Note You might see this attribute if you have a legacy version of Server Suite 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 Server Suite software (versions 3.0.x through 4.1.x).
agentVersion	string	The version of the Server Suite 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.

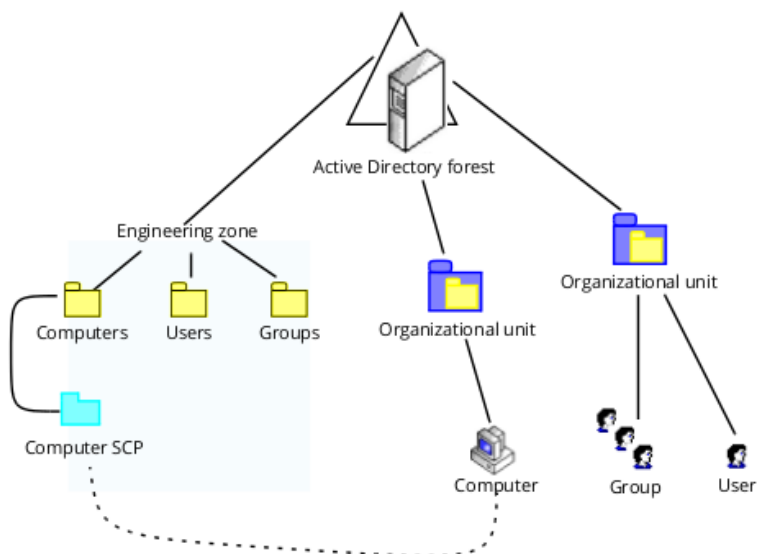
Classic SFU-Compliant Zones (version 3.5)

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.

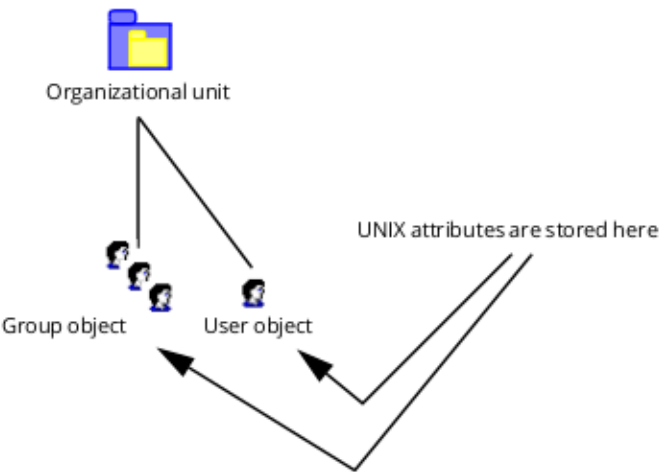
 **Note:** The schema extension must already be installed in the forest. You cannot create SFU-compliant zones if the schema extension is not installed.

Unlike standard Server Suite 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 Server Suite zones and zero or one SFU zone.

The structure of the zone and its sub-containers is the same as the classic Server Suite 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 Server Suite 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 Server Suite zone, except that the zone must also include the NIS domain name and domain attributes. Like the classic Server Suite 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 Server Suite zones](#).

Zone attribute	Stored in Active Directory attribute
ZoneName	cn:ZoneName For example: cn:default
ZoneVersion	displayName:Zoneversion The only valid value is \ \$CimsZoneversion2. For example: displayName:\ \$CimsZoneversion2
Description	description:description:value For example: description:description:Pilot EMEA
NextUid	description:uidnext:value For example: description:uidnext:12098
NextGid	description:gidnext:value For example: description:gidnext:12098
ReservedUids	description:uidreserved:value For example: description:uidreserved:0-99:501
ReservedGids	description:gidreserved:value For example: description:gidreserved:1000-2500

Zone attribute	Stored in Active Directory attribute
Availableshells	description:availableshells:value For example: description:availableshells:/bin/sh
DefaultHomeDirectory	description:defaulthome:value For example: description:defaulthome:/nfs/\${user}
DefaultShell	description:defaultshell:value For example: description:defaultshell:/bin/bash
DefaultGroup	description:defaultgid:value For example: description:defaultgid:12098
ZoneType	description:schema:Dynamic_Schema_version The only valid value for SFU zones is: SFU_3_0 for the Microsoft Services for UNIX (SFU) versions 3.x or 4.x schema extension. For example: description:schema:SFU_3_0
NisDomain	description:Nisdomain:value 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: description: nisdomain:xxx
SFUDomain	description:Sfudomain:value The SFU domain contains the users and groups. The members of an SFU domain can only come from one domain. For example: description:Sfudomain:mfg.ajax.org

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
UnixName	MSSFU30Name:userlogin For example: MSSFU30Name:cain
Uid	MSSFU30UidNumber:value For example: MSSFU30UidNumber:458
Gid	MSSFU30GidNumber:value For example: 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. Delinea 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 zone_bejing to include all users and groups whose NIS domain attribute is set to nisbejing. For example: MSSFU30NisDomain:nisbejing.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. Server Suite 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 zone_beijing to include all users and groups whose NIS domain attribute is set to nisbeijing. For example: MSSFU30NisDomain:nisbeijing.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. Server Suite does not use this attribute, however. Server Suite 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. Server Suite 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 Zones" on page ciii.

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
UnixName	uid:userlogin For example: uid:cain
uid	uidNumber:value For example: uidNumber:458
Gid	gidNumber:value For example: gidNumber:458
Home	unixHomeDirectory:value For example: unixHomeDirectory:/home/shear
Shell	loginShell:value For example: loginShell:/bin/bash
NisDomain	MSSFU30NisDomain:value This attribute must be defined. Server Suite 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 zone_beijing to include all users and groups whose NIS domain attribute is set to nisbeijing. For example: MSSFU30NisDomain:nisbeijing.local
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
UnixName	cn:GroupName For example: cn:performx
Gid	gidNumber:value For example: gidNumber:458
NisDomain	MSSFU30NisDomain:value This attribute must be defined. Server Suite 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 zone_beijing to include all users and groups whose NIS domain attribute is set to nisbeijing. For example: MSSFU30NisDomain:nisbeijing.local
UnixEnabled	Not supported.

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 Server Suite 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 with Commands and Scripts

The OpenLDAP commands provided with the Server Suite 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 Server Suite 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, Server Suite 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 Server Suite 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 Server Suite 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
```

Data Storage for Zones

```
# 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
```

Adding 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: \\$CimsUserVersion3
showInAdvancedViewOnly: TRUE
name: joe
keywords: unix_enabled:True
```

Data Storage for Zones

```
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: \\$cimsUserVersion2
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
```

Zone Requirements

Server Suite 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, Server Suite stores the UNIX-specific information differently depending on the Active Directory schema you are using and the type of Server Suite zones you create.

The Server Suite Windows API provides a logical abstraction of the data model so that you can manipulate Server Suite-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.

Differences between Zone Types

The user and group attributes described in the logical data model are stored differently in Server Suite 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 Server Suite and RFC 2307-compatible zones, user and group UNIX attributes are stored in one `serviceConnectionPoint` object per zone for each user and group.

Schemas and Zones

Server Suite stores UNIX identity data and Server Suite zone data in Active Directory, without modifying or extending the standard Active Directory schema. Server Suite 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 Server Suite 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.

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 Server Suite 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 Server Suite zones (2.x, 3.x, 4.x)	Yes	Yes	No
Classic RFC 2307-compatible	Yes	Yes	No
Hierarchical Server Suite 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 Zone Types" on the previous page.

Supported Zone Types

Standard Server Suite zones use the default data storage model. However, Server Suite 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.

Data Storage for Zones

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 Server Suite 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.

Server Suite Object Reference

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

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:

Method	Description
Commit	Commits changes in the role to Active Directory. (Inherited from <code>[RoleAssignment]</code> (../roleassignment/index.md).
ClearCustomAttributes	VBScript interface to clear the custom attributes for this class. (Inherited from <code>[RoleAssignment]</code> (../roleassignment/index.md).

Method	Description
Delete	Deletes the role. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
GetComputerRole	Returns the computer role that logically contains this role assignment. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
GetTrustee	Returns the trustee being assigned. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
ICustomAttributeContainer GetCustomAttributeContainer	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class. (Inherited from RoleAssignment .)
SetCustomAttribute	VBScript interface to set the custom attributes for this class. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
Validate	Validates this role assignment. (Inherited from [RoleAssignment] (../roleassignment/index.md).)

Properties

The AzRoleAssignment class provides the following properties:

Property	Description
CustomAttributes	VBScript only: Gets or sets custom attributes for this class. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
EndTime	Determines the time at which this role becomes inactive. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
Id	Gets the GUID of the role assignment. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
IsRoleOrphaned	Indicates whether the role assignment is orphaned due to missing or invalid data. (Inherited from RoleAssignment)
IsTrusteeOrphaned	Indicates whether the role assignment is orphaned due to a missing trustee. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
LocalTrustee	Gets the local trustee being assigned. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
Role	Gets the role the trustee is assigned to. (Inherited from [RoleAssignment] (../roleassignment/index.md).)

Property	Description
StartTime	Specifies the time from which this role becomes effective. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
TrusteeDn	Gets the distinguished name of the trustee assigned the role. (Inherited from [RoleAssignment] (../roleassignment/index.md).)
TrusteeType	Gets the trustee type of the role assignment. (Inherited from RoleAssignment .)

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 Delinea 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 Delinea, 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 Delinea 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 Delinea 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:

Method	Description
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 Delinea 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.

Method	Description
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 object.
[GetUserByPath](Returns a user object with its related data by its LDAP path.
GetWindowsUser	Returns a Windows user object.
GetWindowsUserByPath	Returns a Windows user object given the path to the object.
GetZone	Returns a zone object with its related data by object name.
GetZoneByPath	Returns a zone object with its related data by its LDAP path.
IsForestConfigured	Checks whether the forest is properly configured with valid Delinea licenses.
LoadLicenses	Returns all of the Delinea licenses for the connected domain.

Properties

The Cims class provides the following properties:

Property	Description
Password	Gets the password used to establish the connection to the Active Directory domain.
Server	Gets the domain controller computer name used to establish the connection to the Active Directory domain.
UserName	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 Delinea 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
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

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

ConfigureForest

Configures the Active Directory forest to work with Delinea software.

Syntax

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

Parameters

Specify the following parameter when using this method.

Parameter	Description
licenseContainerPath	The LDAP path to the license container holding your Delinea licenses.

Discussion

Your Delinea 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
container	The IADs interface of the parent container object to be used to store the new zone. You can use the standard ADSI GetObject function to retrieve this interface.
name	The name of the new zone.

Return value

The zone object and its related data as `Centrify.DirectControl.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: There is no 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 Schema .
objectType	The Active Directory object type to use for the zone. The valid values are: 0 defines the zone object as a Container object. 1 defines the zone object as an Organization Unit.

Return value

The zone object as Centrify.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 Delinea-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 Delinea-specific information, such as the version of the Delinea 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 Delinea-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 Delinea Windows API creates a *computer zone*, a Delinea 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 Delinea-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 Delinea-specific data.

Syntax

`IGroup GetGroup(IADs group)`

Parameter

Specify the following parameter when using this method:

Parameter	Description
<code>group</code>	The IADs interface to the group object you want to retrieve. You can use the standard ADSI <code>GetObject</code> 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 Delinea-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 Delinea-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\n" +
        "\"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 Delinea-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 Delinea-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 Delinea-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 Delinea-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 Delinea-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 Delinea-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 Delinea 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 Delinea licenses installed on the connected domain.

Syntax

```
ILicensesCollection LoadLicenses()
```

Return value

The `Centrify.DirectControl.API.Licenses` object containing the collection of Delinea 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:

Method	Description
Commit	Commits changes in the right to Active Directory. (Inherited from Right .)
Delete	Removes the right. (Inherited from Right .)

Properties

The Command class provides the following properties:

Property	Description
AllowNestedExecution	If <code>true</code> , allows the command to start another program or open a new shell.
AuthenticationType	Specifies the type of authentication required to run a command.
CommandPattern	Gets or sets the command string that is matched to identify the command.
CommandPatternType	Gets or sets the type of pattern used to match the command.
Description	Gets or sets the description of the right. (Inherited from Right .)
DzdoRunAsGroupList	Gets or sets the list of groups allowed to run this command using <code>dzdo</code> .
DzdoRunAsUserList	Gets or sets the list of users and groups allowed to run this command using <code>dzdo</code> .
DzshRunAsUser	Gets or sets the user this command runs as when executed with <code>dzsh</code> .
Guid	Gets the GUID of the right.
IsReadable	Indicates whether the right is readable. (Inherited from Right .)

Property	Description
IsResetVariables	Resets or removes a default set of environment variables when running the command.
IsWritable	Indicates whether the right is writable. (Inherited from Right .)
MatchPath	Gets or sets the path for matching the specified command name.
Name	Gets or sets the name of the right. (Inherited from Right .)
PreserveGroupMembership	Determines whether to retain the user's group membership while executing a command.
UMask	Gets or sets the UMask value to use for the command.
VariablesToAdd	Gets or sets the list of environment name-value pairs to add, such as var1=a, var2=b, var3=c.
VariablesToKeepOrDelete	Gets or sets the list of environment variables to keep or delete.
Weight	Gets or sets the weight for this command.
Zone	Gets the zone to which this right belongs. (Inherited from Right .)

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 `centrifydc.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 `centrifydc.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:

Method	Description
GetEnumerator	Returns an enumeration of Command objects.

GetEnumerator

Returns an enumeration of Command objects.

Syntax

```
IEnumerator GetEnumerator()
```

Return value

The set of Command objects.

Computer

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

Syntax

```
public interface IComputer
```

Methods

The `Computer` class provides the following methods:

Method	Description
Commit	Commits changes to the computer object and saves them in Active Directory.
Delete	Removes the computer profile from Active Directory.
GetDirectoryEntry	Returns the directory entry for a computer object.
Refresh	Reloads the computer object data from the data in Active Directory.

Properties

The `Computer` class provides the following properties:

Property	Description
<u>AdsInterface</u>	Gets the IADs interface for the computer object from Active Directory.
<u>ADsPath</u>	Gets the LDAP path to the computer object.
<u>AgentVersion</u>	Gets the version number of the Delinea Agent as it is stored in Active Directory.
<u>CanonicalName</u>	Gets the canonical name of the computer object.
<u>IsOrphan</u>	Indicates whether the computer profile has a computer object associated with it.
<u>IsReadable</u>	Indicates whether the computer object is readable.
<u>IsWritable</u>	Indicates whether the computer object is writable.
<u>JBossEnabled</u>	Gets or sets the attribute that enables JBoss access for a computer account.
<u>Name</u>	Gets the computer name of the computer object.
<u>ProfileADsPath</u>	Gets the LDAP path to the computer profile associated with a computer object.
<u>SchemaVersion</u>	Gets the version of the Active Directory schema.
<u>TomcatEnabled</u>	Gets or sets the attribute that enables Tomcat access for a computer account.
<u>Version</u>	Gets the version number of the Active Directory schema.
<u>WebLogicEnabled</u>	Gets or sets the attribute that enables WebLogic access for a computer account.
<u>WebSphereEnabled</u>	Gets or sets the attribute that enables WebSphere access for a computer account.
<u>Zone</u>	Gets the zone associated with the Computer object.
<u>ZoneMode</u>	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");
...
```

AdsIInterface

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

Syntax

```
IADs AdsIInterface {get;}
```

Property value

The IADs interface for the computer object.

Example

The following code sample illustrates using AdsIInterface 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.AdsiInterface.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 Delinea Agent as it is stored in Active Directory.

Syntax

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

Property value

The version number of the Delinea 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 Delinea 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 Delinea 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 Delinea 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
...
```

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
...
```

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 Delinea .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 Delinea 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 Delinea 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 Delinea 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:

Method	Description
Find	Finds the group added to the computer.
GetEnumerator	Returns the enumeration of GroupUnixProfile objects. (Inherited from GroupUnixProfiles .)
Refresh	Reloads the cached GroupUnixProfile objects. (Inherited from GroupUnixProfiles .)

Properties

The `ComputerGroupUnixProfiles` class provides the following properties:

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

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:

Method	Description
AddAccessGroup	Adds a user group to this computer role.
AddRoleAssignment	Adds an empty role assignment.

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

Properties

The ComputerRole class provides the following properties:

Property	Description
CustomAttributes	VBScript only: Gets or sets custom attributes for this computer role.

Property	Description
Description	Gets or sets the description of this computer role.
Group	Gets or sets the AD computer group associated with this computer role.
IsOrphan	Indicates whether this computer role is an orphan.
Name	Gets or sets the name of this computer role.
Zone	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: 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
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 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.

CustomAttributes

Syntax

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

Property value

The custom attribute for this computer role.

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
<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 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 (`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
role	The role for which you want a role assignment.
dn	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
id	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
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.

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
<code>role</code>	The role of the user.

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 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`).

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.

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
name	The name of the custom attribute.
value	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`).

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:

Method	Description
GetEnumerator	Gets the enumerator you can use to enumerate all computer roles.

Properties

The ComputerRoles class provides the following property:

Property	Description
IsEmpty	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:

Method	Description
GetEnumerator	Gets an enumerator you can use to enumerate all computer objects.

Properties

The `ComputerRoles` class provides the following property:

Property	Description
IsEmpty	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:

Method	Description
Find	Finds the user added to the specified computer.
GetEnumerator	Returns the enumeration of user profiles. (Inherited from UserUnixProfiles .)
Refresh	Reloads the cached user UNIX profiles. (Inherited from UserUnixProfiles .)

Properties

The ComputerUserUnixProfiles class provides the following properties:

Property	Description
Count	Gets the number of UserUnixProfile objects in this collection. (Inherited from UserUnixProfiles .)
IsEmpty	Determines whether the collection is empty. (Inherited from UserUnixProfiles .)

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
computer	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.

CustomAttribute

The CustomAttribute class contains a custom attribute. Available for .NET only.

Properties

The CustomAttribute class provides the following properties:

Property	Description
Name	The name of the custom attribute, specified as a string.
value	The value of the custom attribute, specified as a string.

CustomAttributeContainer

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

Methods

The CustomAttributeContainer class provides the following methods:

Method	Description
ClearCustomAttributes	Clears the custom attributes.
ICustomAttributes GetCustomAttributes	Interface to return the custom attributes.
SetCustomAttribute	Interface to set the custom attributes for this class.
ValidateCustomAttributes	Validates the custom attributes.

GetCustomAttributes

Gets the CustomAttributes.

Syntax

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

Parameter

Specify the following parameter when using this method:

Parameter	Description
computer	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:

Method	Description
<code>IEnumerator</code> GetEnumerator	Interface to enumerate the custom attributes.

GetEnumerator

Returns an enumeration of `CustomAttributes` objects.

Syntax

```
IEnumerator GetEnumerator()
```

Return value

The set of `CustomAttributes` objects.

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 Delinea-specific methods and properties for managing the fields in NIS map records. Only the Delinea-specific methods and properties are described in this reference.

Methods

The `Entry` class provides the following Delinea-specific methods:

Method	Description
<code>Clone</code>	Makes a clone of the NIS map entry. Inherited from <code>ICloneable</code> .
Commit	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. Inherited from <code>IDisposable</code> .
GetDirectoryEntry	Returns the <code>DirectoryEntry</code> attribute for the NIS map entry object.

Properties

The Entry class provides the following Delinea-specific properties:

Property	Description
Comment	Gets or sets the comment field associated with a specific key in a map entry.
IsReadable	Indicates whether the map entry is readable.
IsWritable	Indicates whether the map entry is writable.
Key	Gets or sets the key field in a map entry.
Map	Gets the NIS map associated with the map entry.
Value	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$wOrd"
'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\%wOrd"
'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\sw0rd"
'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\sw0rd"
'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\swOrd"
'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\sw0rd"
'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,)"
...
```

Group

Delinea 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:

Method	Description
AddUnixProfile	Adds a new UNIX group profile to a zone.
Commit	Validates and saves changes to the group object in Active Directory.
CommitWithoutCheck	Saves changes to the group object in Active Directory without performing any validation.
GetDirectoryEntry	Returns the directory entry for an Active Directory group object from Active Directory.
GetRoleAssignmentsFromDomain	Returns the collection of all role assignments for a group in a specified domain.
GetRoleAssignmentsFromForest	Returns the collection of all role assignments for a group in a specified forest.
Refresh	Reloads the group object data from the data in Active Directory.

Properties

The Group class provides the following properties:

Property	Description
AdsInterface	Gets the IADs interface for an Active Directory group.
ADsPath	Gets the LDAP path for an Active Directory group.
ID	Gets the unique identifier for an Active Directory group.
UnixProfiles	Gets the GroupUnixProfiles 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
...
```

AdsIInterface

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

Syntax

```
IADsGroup AdsIInterface {get;}
```

Property value

The IADsGroup interface for the group object.

Example

The following code sample illustrates using AdsIInterface 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.AdsIInterface
...
```

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: There is no 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();
}
...
```

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:

Method	Description
Commit	Commits changes to the pending group object and saves them in Active Directory.
Delete	Marks the pending group profile for deletion from Active Directory.
Dispose	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from IDisposable .
GetMembers	Returns the members of a pending import group.
Import	Links the pending import group profile with the specified Active Directory group account.
UpdateStatus	Checks Active Directory for groups that match or conflict with a pending import group.

Properties

The GroupInfo class provides the following properties:

Property	Description
CandidateDN	Gets the distinguished name (DN) of the import candidate.
GID	Gets or sets the UNIX group identifier (GID) for the pending import group profile.
ID	Gets the unique ID of the pending import group object.
IsImported	Indicates whether the pending import group has been successfully imported.
Members	Gets all of a pending import group's members.
Name	Gets or sets the UNIX group name for a pending import group.
Source	Gets the text string that describes the source of the pending import data.
Status	Gets the status of the pending import group.
StatusDescription	Gets a text string that provides detailed information about the status of the pending import group.
TimeStamp	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
group	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/Acme/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:

Method	Description
<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:

Property	Description
<code>Count</code>	Determines the total number of pending import group profiles defined in the collection represented by the <code>GroupInfos</code> object.
<code>IsEmpty</code>	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
id	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:

Method	Description
<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:

Property	Description
CandidateDN	Gets or sets the distinguished name (DN) of the pending import group members.
Name	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:

Method	Description
Add	Adds a new UNIX user as a member of the pending import group.
AddRange	Adds a list of new UNIX users as members of the pending import group.
Clear	Removes all of the group members from a pending import group.
Dispose	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from <code>IDisposable</code> .
GetEnumerator	Returns an enumeration of <code>GroupMember</code> objects. Inherited from <code>IEnumerable</code> .
Remove	Removes the specified group member from the list of members in a pending import group.

Properties

The `GroupMembers` class provides the following properties:

Property	Description
Count	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
<code>memberNames</code>	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.

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:

Method	Description
<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:

Property	Description
<u>ADsPath</u>	Gets the LDAP path to the UNIX group profile.
<u>Cims</u>	Gets the <code>Cims</code> data for the group profile.
<u>Group</u>	Gets the Active Directory group to which the <code>GroupUnixProfile</code> object belongs (Active Directory groups only).
<u>GroupID</u>	Gets or sets the numeric group identifier (GID) for the group profile.
<u>ID</u>	Gets the unique identifier for the UNIX group profile.
<u>IsForeign</u>	Indicates whether the UNIX profile for a group is in a different forest than its corresponding Active Directory group (Active Directory groups only).
<u>IsMembershipRequired</u>	Determines whether an Active Directory group is a required group (Active Directory groups only).
<u>IsOrphan</u>	Indicates whether this UNIX group profile is an orphan (Active Directory groups only).
<u>IsReadable</u>	Indicates whether the Active Directory object is readable.
<u>IsSFU</u>	Indicates whether this UNIX group is an SFU zone profile (Active Directory groups only).

Property	Description
IsWritable	Indicates whether the Active Directory object is writable.
Members	Gets or sets the local group members of the UNIX group profile (local groups only).
Name	Gets or sets the group name of the UNIX group profile.
ProfileState	Gets or sets the profile state of the local group profile (local groups only).
Type	Gets the type of the UNIX group profile.
UnixEnabled	Determines whether the UNIX information is enabled.
Zone	Gets the zone object for the current GroupUnixProfile 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=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 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 `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 Delinea 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
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 Delinea 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
...
```

`[title]: # (Type) [tags]: # (windows api) [priority]: # (21)

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

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

Discussion

The Private group type is only applicable to early versions of Delinea 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:

Method	Description
GetEnumerator	Returns an enumeration of GroupUnixProfile objects.
Refresh	Reloads the GroupUnixProfiles object data from the data in Active Directory.

Properties

The GroupUnixProfiles class provides the following properties:

Property	Description
Count	Gets the total number of UNIX group profiles in the collection of GroupUnixProfiles for an Active Directory group.
IsEmpty	Indicates whether the GroupUnixProfiles 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:

Method	Description
Commit	Commits changes to the <code>GroupUnixProfile</code> object to Active Directory. (Inherited from GroupUnixProfile .)

Method	Description
Delete	Marks the UNIX group profile object for deletion from Active Directory. (Inherited from GroupUnixProfile .)
GetComputer	Returns the computer to which this group profile belongs.
GetDirectoryEntry	Returns the directory entry for a UNIX group profile from Active Directory. (Inherited from GroupUnixProfile .)
InheritFromParent	Clears all property values so that all UNIX attributes for this user are inherited from the parent zone.
Refresh	Reloads the GroupUnixProfile object data from the data in Active Directory. (Inherited from GroupUnixProfile .)
ResolveEffectiveProfile	Resolves the effective profile.
Validate	Validates data in the GroupUnixProfile object before the changes are committed to Active Directory. (Inherited from GroupUnixProfile .)

Properties

The `HierarchicalGroup` class provides the following properties:

Property	Description
ADsPath	Gets the LDAP path to the UNIX group profile. (Inherited from GroupUnixProfile .)
Cims	Gets the Cims data for the group profile. (Inherited from GroupUnixProfile .)
EffectiveGid	Gets the effective GID of the group.
EffectiveIsMembershipRequired	Indicates whether members of this group can remove the group from their currently active set of groups (not applicable to local groups).
EffectiveMembers	Gets members of the local group (local groups only).
EffectiveName	Gets the UNIX name of the group (not applicable to local groups).
EffectiveProfileState	Gets the profile state of the local group (local groups only).

Property	Description
<u>Group</u>	Gets the Active Directory group to which the GroupUnixProfile object belongs (not applicable to local groups). (Inherited from <u>GroupUnixProfile</u> .)
<u>GroupID</u>	Gets or sets the numeric group identifier (GID) for the group profile. (<u>GroupUnixProfile</u> .)
<u>ID</u>	Gets the unique identifier for the UNIX group profile. (Inherited from <u>GroupUnixProfile</u> .)
<u>IsEffectiveGidDefined</u>	Indicates whether there is an effective GID for this group.
<u>IsEffectiveIsMembershipRequiredDefined</u>	Indicates whether there is an effective membership requirement for this group (not applicable to local groups).
<u>IsEffectiveMembersDefined</u>	Indicates whether EffectiveMembers is defined for this group (local groups only).
<u>IsEffectiveNameDefined</u>	Indicates whether there is an effective name for this group.
<u>IsEffectiveProfileStateDefined</u>	Indicates whether there is an effective profile state defined for this group (local groups only).
<u>IsProfileStateDefined</u>	Determines whether the profile state is defined for this group (local groups only).
<u>IsForeign</u>	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 <u>GroupUnixProfile</u> .)
<u>IsGidDefined</u>	Determines whether the GID is defined for this group.
<u>IsMembersDefined</u>	Determines whether Members is defined for this local group (local groups only).
<u>IsMembershipRequired</u>	Determines whether an Active Directory group is a required group (not applicable to local groups). (Inherited from <u>GroupUnixProfile</u> .)
<u>IsMembershipRequiredDefined</u>	Determines whether the membership requirement is defined for this group (not applicable to local groups).
<u>IsNameDefined</u>	Determines whether a name is defined for this group.

Property	Description
<u>IsOrphan</u>	Indicates whether this UNIX group profile is an orphan (not applicable to local groups). (Inherited from <u>GroupUnixProfile</u> .)
<u>IsReadable</u>	Determines whether the Active Directory object is readable. (Inherited from <u>GroupUnixProfile</u> .)
<u>IsSFU</u>	Indicates whether this UNIX group is an SFU zone profile (not applicable to local groups). (Inherited from <u>GroupUnixProfile</u> .)
<u>IsWritable</u>	Determines whether the Active Directory object is writable. (Inherited from <u>GroupUnixProfile</u> .)
<u>Members</u>	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 <u>GroupUnixProfile</u> .)
<u>Name</u>	Gets or sets the group name of the UNIX group profile. (Inherited from <u>GroupUnixProfile</u> .)
<u>ProfileState</u>	Gets or sets the profile state of a local group (local groups only). (Inherited from <u>GroupUnixProfile</u> .)
<u>Type</u>	Gets the type of the UNIX group profile. (Inherited from <u>GroupUnixProfile</u> .)
<u>UnixEnabled</u>	Determines whether the UNIX information is enabled. (Inherited from <u>GroupUnixProfile</u> .)
<u>Zone</u>	
<u>Zone</u>	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:

Method	Description
AddUserRoleAssignment	Returns a new user role assignment.
Commit	Commits changes to the <code>userUnixProfile</code> object to Active Directory. (Inherited from UserUnixProfile .)
Delete	Marks the UNIX user profile object for deletion from Active Directory. (Inherited from UserUnixProfile .)
GetComputer	Returns the computer to which this user profile belongs.

Method	Description
GetDirectoryEntry	Returns the directory entry for a UNIX user profile from Active Directory. (Inherited from userUnixProfile .)
GetEffectiveUserRoleAssignments	Returns the effective user role assignments.
GetPrimaryGroup	Returns the UNIX profile of the primary group of the user. (Inherited from userUnixProfile .)
GetUserRoleAssignment	Returns a user role assignment for this UNIX user.
GetUserRoleAssignments	Returns all the user role assignments for this UNIX user.
InheritFromParent	Clears all property values so that all UNIX attributes for this user are inherited from the parent zone.
Refresh	Reloads the userUnixProfile object data from the data in Active Directory. (Inherited from userUnixProfile .)
ResolveEffectiveProfile	Resolves the effective profile to be used when the user logs on to the computer.
ResolveEffectiveRoles	Resolves the effective roles for this user.
Validate	Validates data in the userUnixProfile object before the changes are committed to Active Directory. (Inherited from userUnixProfile .)

Properties

The `HierarchicalUser` class provides the following properties:

Property	Description
ADsPath	Gets the LDAP path to the UNIX user profile. (Inherited from userUnixProfile .)
Cims	Gets the Cims data for the user profile. (Inherited from userUnixProfile .)
EffectiveGecos	Gets the contents of the effective GECOS field of the user profile.
EffectiveGecosZone	Gets the hierarchical zone of the effective GECOS.
EffectiveHomeDirectory	Gets the effective home directory of the user.

Property	Description
<u>EffectiveHomeDirectoryZone</u>	Gets the zone of the user's home directory.
<u>EffectiveIsUseAutoPrivateGroup</u>	Indicates whether this user uses an auto private group (not applicable to local user profiles).
<u>EffectiveName</u>	Gets the user's effective logon name.
<u>EffectiveNameZone</u>	Gets the zone of the user's effective UNIX name.
<u>EffectivePrimaryGroup</u>	Gets the effective primary group GID of the user.
<u>EffectivePrimaryGroupZone</u>	Gets the zone of the primary group GID.
<u>EffectiveProfileState</u>	Gets the effective profile state of the local user (local user profiles only).
<u>EffectiveProfileStateZone</u>	Gets the zone which defines the effective profile state
<u>EffectiveShell</u>	Gets the effective logon shell of the user.
<u>EffectiveShellZone</u>	Gets the zone of the effective logon shell.
<u>EffectiveUid</u>	Gets the effective UID of the user.
<u>EffectiveUidZone</u>	Gets the zone of the user's effective UID.
<u>Gecos</u>	Gets or sets the contents of the GECOS field explicitly set in the user profile of the current zone.
<u>HomeDirectory</u>	Gets or sets the home directory of the user. (Inherited from <u>UserUnixProfile</u> .)
<u>ID</u>	Gets the unique identifier for the UNIX user profile. (Inherited from <u>UserUnixProfile</u> .)
<u>IsEffectiveGecosDefined</u>	Indicates whether there is an effective GECOS for this user.
<u>IsEffectiveHomeDirectoryDefined</u>	Indicates whether there is an effective home directory defined for this user.
<u>IsEffectiveNameDefined</u>	Indicates whether there is an effective name for this user.
<u>IsEffectivePrimaryGroupDefined</u>	Indicates whether a primary group is defined for this user.

Property	Description
<u>IsEffectiveProfileStateDefined</u>	Indicates whether there is an effective profile state for this local user (local user profiles only).
<u>IsEffectiveShellDefined</u>	Indicates whether there is an effective shell defined for this user.
<u>IsEffectiveUidDefined</u>	Indicates whether the user has an effective UID.
<u>IsEffectiveUseAutoPrivateGroupDefined</u>	Indicates whether the auto private group flag is defined for this user (not applicable to local user profiles).
<u>IsForeign</u>	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 <u>UserUnixProfile</u> .)
<u>IsGecosDefined</u>	Determines whether the GECOS is defined in this profile.
<u>IsHomeDirectoryDefined</u>	Determines whether the home directory is defined in this profile.
<u>IsNameDefined</u>	Determines whether a name is defined in this profile.
<u>IsOrphan</u>	Indicates whether this UNIX user profile is an orphan (not applicable to local user profiles). (Inherited from <u>UserUnixProfile</u> .)
<u>IsPrimaryGroupDefined</u>	Determines whether there is a GID defined for this user in this zone.
<u>IsProfileStateDefined</u>	Gets or sets whether the profile state is defined in this local user profile (local user profiles only).
<u>IsReadable</u>	Determines whether the Active Directory object is readable. (Inherited from <u>UserUnixProfile</u> .)
<u>IsSecondary</u>	Indicates whether this is a secondary profile (not applicable to local user profiles).
<u>IsSFU</u>	Indicates whether this user object uses the Microsoft Services for UNIX (SFU) schema extension (not applicable to local user profiles). (Inherited from <u>UserUnixProfile</u> .)
<u>IsShellDefined</u>	Determines whether the shell is defined in this profile.

Property	Description
<u>IsUidDefined</u>	Determines whether the ID is defined in this profile.
<u>IsUseAutoPrivateGroup</u>	Determines whether this user uses auto private groups (not applicable to local user profiles).
<u>IsUseAutoPrivateGroupDefined</u>	Determines whether the auto private group flag is defined (not applicable to local user profiles).
<u>IsWritable</u>	Determines whether the Active Directory object is writable. (Inherited from <u>UserUnixProfile</u> .)
<u>Name</u>	Gets or sets the user name of the UNIX user profile. (Inherited from <u>UserUnixProfile</u> .)
<u>PrimaryGroup</u>	Gets or sets the GID of the user's primary group. (Inherited from <u>UserUnixProfile</u> .)
<u>ProfileState</u>	Gets or sets the profile state of a local user profile (local user profiles only). (Inherited from <u>UserUnixProfile</u> .)
<u>Shell</u>	Gets or sets the user's default shell. (Inherited from <u>UserUnixProfile</u> .)
<u>Type</u>	Gets the type of the UNIX user profile. (Inherited from <u>UserUnixProfile</u> .)
<u>UnixEnabled</u>	Determines whether the UNIX information is enabled. (Inherited from <u>UserUnixProfile</u> .)
<u>User</u>	Gets the user to whom this UNIX profile belongs (not applicable to local user profiles). (Inherited from <u>UserUnixProfile</u> .)
<u>UserId</u>	Gets or sets the user identifier (UID) for the user profile. (Inherited from <u>UserUnixProfile</u> .)
<u>Zone</u>	

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](../computerrole/commit.md)` 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: There is no 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 assignment 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
zone	The zone for which you want the group and user role assignments.
computer	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 GID 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 GID 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:

Method	Description
AddAccessGroup	Adds an empty role assignment to a group
AddComputerRole	Creates a computer role under this zone.
AddGroupPartialProfile	Adds a partial profile for a specified group.

Method	Description
<u>AddLocalGroupPartialProfile</u>	Adds a partial profile for a specified local group.
<u>AddLocalUserPartialProfile</u>	Adds a partial profile for a specified local user.
<u>AddMitUser</u>	Adds an MIT Kerberos realm trusted user to this zone. (Inherited from <u>Zone</u> .)
<u>AddRoleAssignment</u>	Adds an empty role assignment.
<u>AddUserPartialProfile</u>	Adds a partial profile for a specified user.
<u>Commit</u>	Commits changes to the group object to Active Directory. (Inherited from <u>Zone</u> .)
<u>CreateCommand</u>	Creates a command right for the zone.
<u>CreateImportPendingGroup</u>	Creates a pending imported group in this zone. (Inherited from <u>Zone</u> .)
<u>CreateImportPendingUser</u>	Creates a pending imported user in this zone. (Inherited from <u>Zone</u> .)
<u>CreateNetworkAccess</u>	Creates a network application access right.
<u>CreatePamAccess</u>	Creates a PAM application access right.
<u>CreateRole</u>	Creates a role in the zone.
<u>CreateSshRight</u>	Creates an SSH application access right.
<u>CreateWindowsApplication</u>	Creates a Windows application access right.
<u>CreateWindowsDesktop</u>	Creates a Windows Desktop access right.
<u>Delete</u>	Marks the zone for deletion from Active Directory. (Inherited from <u>Zone</u> .)
<u>GeneratePredefinedRights</u>	Generates predefined SSH and PAM rights in this zone.
<u>GeneratePredefinedRoles</u>	Generates predefined user roles in this zone.
<u>GetAccessGroup</u>	Returns a group assigned to this zone given a role for the group.
<u>GetAccessGroups</u>	Returns an enumeration of groups in the zone.
<u>GetChildZones</u>	Returns an enumeration of this zone's child zones.

Method	Description
<u>GetCommand</u>	Returns the privileged command right with a specific name or GUID.
<u>GetCommands</u>	Returns an enumeration of all the privileged command rights in the zone.
<u>GetComputerByDN</u>	Returns the computer profile in the zone given the distinguished name of the profile. (Inherited from <u>Zone</u> .)
<u>GetComputerRole</u>	Returns a specific computer role under this zone.
<u>GetComputerRoles</u>	Returns an enumeration of all the computer roles under this zone.
<u>GetComputers</u>	Returns an enumeration of all the computers in the zone. (Inherited from <u>Zone</u> .)
<u>GetComputersContainer</u>	Returns the Active Directory object for the Computers node. (Inherited from <u>Zone</u> .)
<u>GetDirectoryEntry</u>	Returns the Active Directory object for the zone. (Inherited from <u>Zone</u> .)
<u>GetDisplayName</u>	Returns the display name of this zone. (Inherited from <u>Zone</u> .)
<u>GetEffectiveCommands</u>	Returns all the command rights that can be assigned to users in the zone, including inherited rights.
<u>GetEffectiveNetworkAccesses</u>	Returns all the network access rights that can be assigned to users in the zone, including inherited rights.
<u>GetEffectivePamAccesses</u>	Returns all the PAM application access rights that can be assigned to users in the zone, including inherited rights.
<u>GetEffectiveRoles</u>	Returns all the user roles that can be assigned to users in the zone, including inherited roles.
<u>GetEffectiveSshs</u>	Returns all the SSH application access rights that can be assigned to users in the zone, including inherited rights.
<u>GetEffectiveUserUnixProfiles</u>	Returns an enumeration of effective users under this zone.
<u>GetEffectivewindowsApplications</u>	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.

Method	Description
<u>GetEffectivewindowsDesktops</u>	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.
<u>GetEffectivewindowsUsers</u>	Returns all the Windows users in the zone, including users inherited from zones higher in the hierarchy.
<u>GetLocalGroupsContainer</u>	Returns the DirectoryEntry of the local groups container. (Inherited from <u>Zone</u> .)
<u>GetLocalUserUnixProfile</u>	Returns the local UNIX group profile for a specified group name in the zone. (Inherited from <u>Zone</u> .)
<u>GetLocalUserUnixProfileByDN</u>	Returns a local group profile using the distinguished name (DN) of the profile. (Inherited from <u>Zone</u> .)
<u>GetLocalGroupUnixProfileByGid</u> (Int32)	Returns the local group profile using the Group Identifier (GID). This method is exposed to the .COM interface. (Inherited from <u>Zone</u> .)
<u>GetLocalGroupUnixProfiles</u>	Returns a list of the local group profiles in the zone. (Inherited from <u>Zone</u> .)
<u>GetLocalUsersContainer</u>	Returns the directory entry of the local users container. (Inherited from <u>Zone</u> .)
<u>GetLocalUserUnixProfile</u>	Returns the local user profile using the specified user name. (Inherited from <u>Zone</u> .)
<u>GetLocalUserUnixProfileByDN</u>	Returns the local user profile specified by the distinguished name (DN) of the profile. (Inherited from <u>Zone</u> .)
<u>GetLocalUserUnixProfileByUid</u> (Int32)	Returns the local user profile using the User Identifier (UID). This method is exposed to the .COM interface (Inherited from <u>Zone</u> .)
<u>GetLocalUserUnixProfiles</u>	Returns a list of the local user profiles in the zone. (Inherited from <u>Zone</u> .)
<u>GetNetworkAccess</u>	Returns the specified network access right.
<u>GetNetworkAccesses</u>	Returns all the network access rights that can be assigned to users in the zone.
<u>GetGroupsContainer</u>	Returns the Active Directory object for the Groups container. (Inherited from <u>Zone</u> .)

Method	Description
<u>GetGroupUnixProfile</u>	Returns the UNIX group profile in this zone for the specified Active Directory group. (Inherited from <u>Zone</u> .)
<u>GetGroupUnixProfileByDN</u>	Returns the UNIX group profile in this zone for the Active Directory group specified by distinguished name. (Inherited from <u>Zone</u> .)
<u>GetGroupUnixProfileByName</u>	Returns the UNIX group profile in this zone for the Active Directory group specified by group name. (Inherited from <u>Zone</u> .)
<u>GetGroupUnixProfiles</u>	Returns an enumeration of the UNIX groups in the zone. (Inherited from <u>Zone</u> .)
<u>GetImportPendingGroup</u>	Returns the group with the specified ID pending import. (Inherited from <u>Zone</u> .)
<u>GetImportPendingGroups</u>	Returns an enumeration of groups pending import to this zone. (Inherited from <u>Zone</u> .)
<u>GetImportPendingUser</u>	Returns the user with the specified ID pending import. (Inherited from <u>Zone</u> .)
<u>GetImportPendingUsers</u>	Returns an enumeration of users pending import to this zone. (Inherited from <u>Zone</u> .)
<u>GetNetworkAccess</u>	VBScript interface to access NSS variables.
<u>GetNSSVariables</u>	VBScript interface to obtain all NSS variable names.
<u>GetPamAccess</u>	Returns the PAM application access right with the specified name.
<u>GetPamAccesses</u>	Returns an enumeration of all the PAM application rights in the zone.
<u>GetPrimaryUser</u>	Returns the primary profile for the specified user.
<u>GetRole</u>	Returns the role with the specified name or GUID.
<u>GetRoleAssignment</u>	Returns the role assignment for the specified role and trustee.
<u>GetRoleAssignmentById</u>	Returns the role assignment for the specified GUID.
<u>GetRoleAssignments</u>	Returns an enumeration of all the role assignments in the zone.
<u>GetRoleAssignmentToAllADUsers</u>	Returns the role assignment given to all Active Directory users who have a specified role.

Method	Description
<u>GetRoleAssignmentToAllUnixUsers</u>	Returns the role assignment given to all UNIX users who have a specified role.
<u>GetRoles</u>	Returns an enumeration of all the roles in the zone.
<u>GetSecondaryUsers</u>	Returns an enumeration of the secondary profiles for the specified user.
<u>GetSshRight</u>	Returns the SSH application access right with the specified name.
<u>GetSshRights</u>	Returns an enumeration of all the SSH application rights in the zone.
<u>GetSubTreeRoleAssignments</u>	Returns all role assignments under this zone, including role assignments for computer roles and computers.
<u>GetUserProfiles</u>	Returns an enumeration of all the user profiles for the specified user.
<u>GetUserRoleAssignments</u>	Returns an enumeration of all the user role assignments in the zone.
<u>GetWindowsApplication</u>	Returns the specified Windows application right.
<u>GetWindowsApplications</u>	Returns all the Windows application rights in the zone.
<u>GetWindowsComputers</u>	Returns all the Windows computers in the zone.
<u>GetWindowsDesktop</u>	Returns the specified Windows desktop right.
<u>GetWindowsDesktops</u>	Returns all the Windows desktop rights in the zone.
<u>GetUsersContainer</u>	Returns the directory entry of the Users container. (Inherited from <u>Zone</u> .)
<u>GetUserUnixProfileByDN</u>	Returns the UNIX user profile in this zone for the user specified by distinguished name. (Inherited from <u>Zone</u> .)
<u>GetUserUnixProfileByName</u>	Returns the UNIX user profile in this zone for the user specified by user name. (Inherited from <u>Zone</u> .)
<u>GetUserUnixProfiles</u>	Returns an enumeration of all the UNIX user profiles in the zone. (Inherited from <u>Zone</u> .)
<u>GroupUnixProfileExists</u>	Indicates whether the group has a profile in this zone. (Inherited from <u>Zone</u> .)
<u>LocalGroupUnixProfileExists</u>	Indicates whether a UNIX profile exists in the zone for the specified local group. (Inherited from <u>Zone</u> .)

Method	Description
LocalUserUnixProfileExists	Indicates whether a UNIX profile exists in the zone for the specified local user. (Inherited from Zone .)
PrecreateComputerZone	Adds a computer zone to a computer object in this zone.
Refresh	Refreshes the data in this object instance from the data stored in Active Directory. (Inherited from Zone .)
SetNSSVariable	VBScript interface to set the values of NSS variables.
UserUnixProfileExists	Indicates whether the specified user has a profile in this zone. (Inherited from Zone .)

Properties

The HierarchicalZone class provides the following properties:

Property	Description
AdsiInterface	Gets the IADs interface of the zone object in Active Directory. (Inherited from Zone .)
ADsPath	Gets the LDAP path to the zone object. (Inherited from Zone .)
AgentlessAttribute	Gets or sets the attribute used to store the password hash for an agentless client. (Inherited from Zone .)
AvailableShells	Gets or sets an enumeration of available user login shells. (Inherited from Zone .)
Cims	Gets the Cims object managing this zone. (Inherited from Zone .)
DefaultGroup	Gets or sets the default group for new users. (Inherited from Zone .)
DefaultHomeDirectory	Gets or sets the default login directory for new users. (Inherited from Zone .)
DefaultShell	Gets or sets the default login shell for new users. (Inherited from Zone .)
DefaultValueZone	Gets or sets the zone to use for default zone values. (Inherited from Zone .)
Description	Gets or sets the description of the zone. (Inherited from Zone .)

<u>FullName</u>	Gets or sets the full name of the zone. (Inherited from <u>Zone</u> .)
<u>GroupAutoProvisioningEnabled</u>	Indicates whether auto-provisioning of group profiles is enabled for the zone. (Inherited from <u>Zone</u> .)
<u>GroupDefaultName</u>	Gets or sets the default group name.
<u>ID</u>	Gets the unique identifier for the zone. (Inherited from <u>Zone</u> .)
<u>IsChild</u>	Indicates whether this is a child zone.
<u>IsGroupDefaultNameDefined</u>	Indicates whether the group default name is defined.
<u>IsHierarchical</u>	Indicates whether this is a hierarchical zone. (Inherited from <u>Zone</u> .)
<u>IsNextGidDefined</u>	Gets or sets whether Next GID value is configured for this zone.
<u>IsNextUidDefined</u>	Gets or sets whether Next UID value is configured for this zone.
<u>IsReadable</u>	Indicates whether this zone object in Active Directory is readable with the current user credentials. (Inherited from <u>Zone</u> .)
<u>IsSFU</u>	Indicates whether the zone uses the Microsoft Services for UNIX (SFU) schema extension. (Inherited from <u>Zone</u> .)
<u>IsTruncateName</u>	Indicates whether this is a TruncateName zone. (Inherited from <u>Zone</u> .)
<u>IsUseAutoPrivateGroupDefined</u>	Determines whether the UseAutoPrivateGroup flag is defined.
<u>IsUserDefaultGecosDefined</u>	Determines whether the user default GECOS is defined in this profile.
<u>IsUserDefaultHomeDirectoryDefined</u>	Determines whether the user default home directory is defined in this profile.
<u>IsUserDefaultNameDefined</u>	Determines whether the user default name is defined in this profile.
<u>IsUserDefaultPrimaryGroupDefined</u>	Determines whether the user default primary group is defined in this profile.
<u>IsUserDefaultRoleDefined</u>	Determines whether the user default role is defined in this profile.
<u>IsUserDefaultShellDefined</u>	Determines whether the user default login shell is defined in this profile.

<u>Iswritable</u>	Indicates whether this zone object is writable using the provided credential. (Inherited from <u>Zone</u> .)
<u>Licenses</u>	Gets or sets the license container for the zone. (Inherited from <u>Zone</u> .)
<u>MasterDomainController</u>	Gets or sets the master domain controller for the zone. (Inherited from <u>Zone</u> .)
<u>MustMaintainADGroupMembership</u>	Indicates whether Active Directory group membership must be maintained. (Inherited from <u>Zone</u> .)
<u>Name</u>	Gets or sets the name of the zone. (Inherited from <u>Zone</u> .)
<u>NextAvailableGID</u>	Gets or sets the next GID to be used when adding a group (32-bit for COM programs). (Inherited from <u>Zone</u> .)
<u>NextAvailableUID</u>	Gets or sets the next UID to be used when adding a user (32-bit for COM programs). (Inherited from <u>Zone</u> .)
<u>NextGID</u>	Gets or sets the next GID to be used when adding a group (64-bit for .NET modules). (Inherited from <u>Zone</u> .)
<u>NextUID</u>	Gets or sets the next UID to be used when adding a user (64-bit for .NET modules). (Inherited from <u>Zone</u> .)
<u>NISDomain</u>	Gets or sets the NIS domain associated with this SFU zone. (Inherited from <u>Zone</u> .)
<u>NssVariables</u>	Gets the map of profile variables.
<u>Parent</u>	Gets or sets the parent of this zone.
<u>ReservedGID</u>	Gets or sets the list of GIDs not to be used when adding groups. (Inherited from <u>Zone</u> .)
<u>ReservedUID</u>	Gets or sets the list of UIDs not to be used when adding users. (Inherited from <u>Zone</u> .)
<u>Schema</u>	Gets the schema of the zone. (Inherited from <u>Zone</u> .)
<u>SFUDomain</u>	Gets or sets the Active Directory domain associated with this SFU zone for retrieving SFU information. (Inherited from <u>Zone</u> .)

<u>UseAppleGid</u>	Determines whether to use the Apple algorithm to automatically generate the GID when adding a group. The Apple algorithm is based on the globally unique identifier (GUID) for the object.
<u>UseAppleUid</u>	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.
<u>UseAutoGid</u>	Determines whether to use the Delinea algorithm to automatically generate the GID when adding a group. The Delinea algorithm is based on the security identifier (SID) for the object.
<u>UseAutoPrivateGroup</u>	Determines whether this zone defaults to use an auto private group when adding a zone user.
<u>UseAutoUid</u>	Determines whether to use the Delinea algorithm to automatically generate the UID when adding a user. The Delinea algorithm is based on the security identifier (SID) for the object.
<u>UseNextGid</u>	Determines whether to use the NextGID property when adding a group.
<u>UseNextUid</u>	Determines whether to use the NextUID property when adding a user.
<u>UserAutoProvisioningEnabled</u>	Indicates whether auto-provisioning of user profiles is enabled for the zone. (Inherited from <u>Zone</u> .)
<u>UserDefaultGecos</u>	Gets or sets the default GECOS field for new user profiles.
<u>UserDefaultGid</u>	Gets or sets the user default GID when adding a new user profile.
<u>UserDefaultName</u>	Gets or sets the default user name for a new user profile.
<u>UserDefaultPrimaryGroup</u>	Gets or sets the user default GID for new user profiles; for use in VBScript scripts.
<u>UserDefaultRole</u>	Gets or sets the default role for a new user profile.
<u>Version</u>	Gets the version number of the data schema. (Inherited from <u>Zone</u> .)

AddAccessGroup

Adds an empty role assignment to a group.

Syntax

```
IHRoleAssignment 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
groupDE	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.

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

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.

Parameter	Description
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();
}
...
```

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`.

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](../computerrole/commit.md)` method.

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`.

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: There is no 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.GetStringSid(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("windowsDesktop " + 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
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 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 (`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 [GetRoleAssignmentfor](#) 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 `ArgumentNullException` 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
role	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
role	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 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.
usersSr	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 `GetSecondaryUser(DirectoryEntry userDE)` and `GetSecondaryUser(SearchResult usersSr)` 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
userDE	The directory entry for the user for which you want the 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 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)
IHierarchicalZoneComputer PrecreateComputerZone (string dnsName, DirectoryEntry trustee, bool
skipPermissionSetting);
```

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.
<code>skipPermissionSetting</code>	Specifies if permission delegation is skipped when precreating computer zones.

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

Delinea 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

Delinea 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 Delinea 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`, Delinea 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`, Delinea uses the Apple algorithm to generate the numeric user identifier (UID) when adding a user.

UseAutoGid

Determines whether to use the Delinea algorithm to automatically generate the GID when adding a group to the zone. An auto-generated GID when adding a group to the zone. The Delinea 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`, Delinea automatically generates a GID 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`, Delinea 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 Delinea algorithm to automatically generate the UID when adding a user to the zone. The Delinea 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`, Delinea 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`, Delinea 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`, Delinea 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, Delinea 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, Delinea 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, Delinea 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:

Method	Description
<u>AddAccessGroup</u>	Adds a group to the computer.
<u>AddGroupPartialProfile</u>	Adds a computer-specific partial profile for a specified group.
<u>AddLocalGroupPartialProfile</u>	Adds a computer-specific partial profile for a specified local group.
<u>AddLocalUserPartialProfile</u>	Adds a computer-specific partial profile for a specified user.
<u>AddRoleAssignment</u>	Adds an empty role assignment.
<u>AddUserPartialProfile</u>	Adds a computer-specific partial profile for a specified user.
<u>Commit</u>	Commits changes to the group object to Active Directory. (Inherited from <u>Computer</u> .)
<u>CreateImportPendingGroup</u>	Creates a pending imported group in this computer.
<u>CreateImportPendingUser</u>	Creates a pending imported user in this computer.
<u>Delete</u>	Deletes the computer profile from Active Directory. (Inherited from <u>Computer</u> .)
<u>DeleteAllProfiles</u>	Deletes all computer-specific users and groups.
<u>DeleteZone</u>	Deletes the computer zone object if it exists.
<u>GetAccessGroup</u>	Returns a group given a role for the group.
<u>GetAccessGroups</u>	Returns an enumeration of groups in the computer object.
<u>GetDirectoryEntry</u>	Returns the Active Directory object for the computer. (Inherited from <u>Computer</u> .)
<u>GetEffectiveUserUnixProfiles</u>	Returns an enumeration of effective users under this computer zone.
<u>GetGroupUnixProfile</u>	Returns the UNIX group profile in this computer zone for the specified Active Directory group.

Method	Description
<u>GetGroupUnixProfileByDN</u>	Returns the UNIX group profile in this computer zone for the Active Directory group specified by distinguished name.
<u>GetGroupUnixProfileByName</u>	Returns the UNIX group profile in this computer zone for the Active Directory group specified by group name.
<u>GetGroupUnixProfiles</u>	Returns an enumeration of the UNIX groups in this computer zone.
<u>GetImportPendingGroup</u>	Returns the group with the specified ID pending import.
<u>GetImportPendingGroups</u>	Returns an enumeration of groups pending import to this computer zone.
<u>GetImportPendingUser</u>	Returns the user with the specified ID pending import.
<u>GetImportPendingUsers</u>	Returns an enumeration of users pending import to this computer zone.
<u>GetIPendingGroupID</u>	Returns the numeric identifier for the pending import group with the specified group name.

Method	Description
<u>GetLocalGroupUnixProfile</u>	Returns the local UNIX group profile for a specified group name in the zone.
<u>GetLocalUserUnixProfileByDN</u>	Returns a local group profile using the distinguished name (DN) of the profile.
<u>GetLocalGroupUnixProfileByGid</u>	Returns the local group profile using the Group Identifier (GID). This method is exposed to the .COM interface.

Method	Description
<u>GetLocalGroupUnixProfiles</u>	Returns a list of the local group profiles in the zone.
<u>GetLocalUserUnixProfile</u>	Returns the local user profile using the specified user name.
<u>GetLocalUserUnixProfileByDN</u>	Returns the local user profile specified by the distinguished name (DN) of the profile.
<u>GetLocalUserUnixProfileByUid</u>	Returns the local user profile using the User Identifier (UID). This method is exposed to the .COM interface
<u>GetLocalUserUnixProfiles</u>	Returns a list of the local user profiles in the zone.
<u>GetIPendingUserID</u>	Returns the numeric identifier for the pending import user with the specified user name.

Method	Description
<u>GetNssVariable</u>	VBScript interface to access NSS variables.
<u>GetNSSVariables</u>	VBScript interface to obtain all NSS variable names.
<u>GetPrimaryUser</u>	Returns the primary profile for the specified user.
<u>GetRoleAssignment</u>	Returns the role assignment for the specified role and trustee.
<u>GetRoleAssignmentById</u>	Returns the role assignment for the specified GUID.
<u>GetRoleAssignments</u>	Returns the collection of role assignments in the computer.
<u>GetRoleAssignmentToAllADUsers</u>	Returns the role assignment given to all Active Directory users who have a specified role.

Method	Description
<u>GetRoleAssignmentToAllUnixUsers</u>	Returns the role assignment given to all UNIX users who have a specified role.
<u>GetSecondaryUsers</u>	Returns an enumeration of the secondary profiles for the specified user.
<u>GetUserProfiles</u>	Returns an enumeration of all the user profiles for the specified user.
<u>GetUserRoleAssignments</u>	Returns an enumeration of all the user role assignments in this computer zone.
<u>GetUserUnixProfile</u>	Returns the UNIX user profile in this computer zone for the specified user.
<u>GetUserUnixProfileByDN</u>	Returns the UNIX user profile in this computer zone for the user specified by distinguished name.

Method	Description
GetUserUnixProfileByName	Returns the UNIX user profile in this computer zone for the user specified by user name.
GetUserUnixProfileByUid	Returns the UNIX user profile in this computer zone for the user specified by UID.
GetUserUnixProfiles](getuserunixprofiles.md) Returns an enumeration of all the UNIX user profiles in this computer zone. [GroupUnixProfileExists](groupunixprofileexists.md) Indicates whether the group has a profile in this computer zone. [LocalGroupUnixProfileExists](localgroupunixprofileexists.md) Indicates whether a UNIX profile exists in the zone for the specified local group. [LocalUserUnixProfileExists](localuserunixprofileexists.md) Indicates whether a UNIX profile exists in the zone for the specified local user. [Refresh](../computer/refresh.md) Refreshes the data in this object instance from the data stored in Active Directory. (Inherited from [Computer](../computer/index.md) .) [SetNSSVariable](setnssvariable.md) VBScript interface to set the values of NSS variables. [UserUnixProfileExists]	Indicates whether the specified user has a profile in this computer zone.

Properties

The `HierarchicalZoneComputer` class provides the following properties:

Property	Description
AdsiInterface	Gets the IADs interface of the zone object in Active Directory. (Inherited from Computer .)
ADsPath	Gets the LDAP path to the zone object. (Inherited from Computer .)
AgentVersion	Gets the Active Directory client version number. (Inherited from Computer .)
CanonicalName	Gets the canonical name of the computer object. (Inherited from Computer .)
ComputerZoneADsPath	Gets the LDAP path of the computer zone object.

Property	Description
<u>IsOrphan</u>	Indicates whether the CIMs data associated with this object is orphaned by the current credentials. (Inherited from <u>Computer.</u>)
<u>IsOrphanZone</u>	Indicates whether this computer is an orphan zone object.
<u>IsReadable</u>	Indicates whether the CIMS data associated with this object is readable with the current user credentials. (Inherited from <u>Computer.</u>)
<u>IsWritable</u>	Indicates whether the CIMS data associated with this object is writable with the current user credentials. (Inherited from <u>Computer.</u>)
<u>JBossEnabled</u>	Determines whether the computer is enabled for JBoss. (Inherited from <u>Computer.</u>)
<u>Name</u>	Gets or sets the name of the computer object. (Inherited from <u>Computer.</u>)
<u>NssVariables</u>	Gets the map of profile variables.
<u>ProfileADsPath</u>	Gets the LDAP path to the computer UNIX profile. (Inherited from <u>Computer.</u>)
<u>SchemaVersion</u>	Gets the version of the data schema. (Inherited from <u>Computer.</u>)
<u>TomcatEnabled</u>	Determines whether the computer is enabled for Tomcat. (Inherited from <u>Computer.</u>)
<u>UserHomeDirectory</u>	Gets or sets the UNIX directory path that is used to substitute for %{home} in user profiles.
<u>UserShell</u>	Gets or sets the shell that is used to substitute for %{shell} in user profiles.
<u>Version</u>	Gets the version number of the data schema. (Inherited from <u>Computer.</u>)
<u>webLogicEnabled</u>	Determines whether the computer is enabled for WebLogic. (Inherited from <u>Computer.</u>)
<u>webSphereEnabled</u>	Determines whether the computer is enabled for WebSphere. (Inherited from <u>Computer.</u>)
<u>Zone</u>	Gets or sets the zone that this computer joins.
<u>ZoneMode</u>	Gets the zone mode of the computer. (Inherited from <u>Computer.</u>)

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
groupDE	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.

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
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

When you assign computer-level overrides for user, group, or computer role assignments, Delinea 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.

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](../computerrole/commit.md)` 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 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.

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, Delinea 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.

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, Delinea 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`.

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
source	The location of the source data for the group to be imported.
timestamp	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 ([Delete](#)) 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, Delinea 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
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 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 [GetRoleAssignmentfor](#) 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

```
IIHierarchicalGroup 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
dn	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
name	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
id	The GUID of the group that's pending import.
storePendingAD	Specify true if the group is being imported from Active Directory. Specify false if the group is being imported from an XML file.
storePendingFilePath	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.

Parameter	Description
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.
usersSr	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
user	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 `Centrify.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
dn	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
<code>name</code>	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
uid	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 Delinea 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.

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, Delinea 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.

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
userName	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`.

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 `GetNSSVariable` property for VBScript.

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. Pass <code>null</code> to remove the variable.

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.

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
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 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.

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:

Method	Description
<u>ClearCustomAttributes</u>	VBScript interface to clear the custom attributes for this class. Inherited from <u>(RoleAssignment.)</u>
<u>Commit</u>	Commits changes in the role assignment to Active Directory. Inherited from <u>(RoleAssignment.)</u>
<u>Delete</u>	Deletes the role. Inherited from <u>(RoleAssignment.)</u>
<u>ICustomAttributeContainer</u> <u>GetCustomAttributeContainer</u>	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class. Inherited from <u>(RoleAssignment.)</u>
<u>GetTrustee</u>	Returns the trustee being assigned. Inherited from <u>(RoleAssignment.)</u>
<u>SetCustomAttribute</u>	VBScript interface to set the custom attributes for this class. Inherited from <u>(RoleAssignment.)</u>
<u>Validate</u>	Validates the role assignment. Inherited from <u>(RoleAssignment.)</u>

Properties

The `HZRoleAssignment` class provides the following properties:

Property	Description
<u>CustomAttributes</u>	VBScript only: Gets or sets custom attributes for this class. Inherited from <u>(RoleAssignment.)</u>
<u>EndTime</u>	Determines the time at which this role becomes inactive. Inherited from <u>(RoleAssignment.)</u>
<u>Id</u>	Gets the GUID of the role assignment. Inherited from <u>(RoleAssignment.)</u>
<u>IsRoleOrphaned</u>	Indicates whether the role assignment is orphaned due to a missing or invalid role. Inherited from <u>(RoleAssignment.)</u>
<u>IsTrusteeOrphaned</u>	Indicates whether the role assignment is orphaned due to a missing or invalid trustee. Inherited from <u>(RoleAssignment.)</u>

Property	Description
LocalTrustee	Gets the local trustee being assigned. Inherited from (RoleAssignment.)
Role	Gets the role the trustee is assigned to. Inherited from (RoleAssignment.)
StartTime	Specifies the time from which this role becomes effective. Inherited from (RoleAssignment.)
TrusteeDn	Gets the distinguished name of the trustee assigned this role. Inherited from (RoleAssignment.)
TrusteeType	Gets the trustee type of the role assignment. Inherited from (RoleAssignment.)
Zone	

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:

Method	Description
GetTrustee	Returns the user associated with the role.

Properties

The `InheritedRoleAsg` class provides the following properties:

Property	Description
EndTime	Determines the time at which this role becomes inactive.
IsRoleOrphaned	Indicates whether the role assignment is orphaned due to missing or invalid data.
IsTrusteeOrphaned	Indicates whether the role assignment is orphaned due to a missing trustee.
Role	Gets the role the trustee is assigned to.
Source	Gets the role assignment that is the source for this inherited role assignment.
StartTime	Specifies the time from which this role becomes effective.
TrusteeDn	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 Delinea. 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:

Property	Description
Count	Gets the number of licenses provided by a specific key.
ExpiryDate	Gets the date a specific license key is set to expire.
IsValid	Determines whether the license key is an evaluation license.
IsValid	Determines whether the license key being checked is a valid license.
SerialNumber	Gets the serial number of the license key.
Type	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 Delinea 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:

Method	Description
Add	Adds a license key to the set.
GetEnumerator	Returns an enumeration of key objects.
Remove	Removes a license key from the set.

Properties

The keys class provides the following properties:

Property	Description
Count	Gets the number of license keys stored in the set.
Item	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
key	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 Delinea 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:

Property	Description
<u>Count</u>	Determines the total number of licenses of a particular type.
<u>IsEval</u>	Determines whether the license is an evaluation license.
<u>Keys</u>	Gets the license keys associated with the license object.
<u>Type</u>	Gets the license type for the license object.
<u>UsedCount</u>	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 Delinea 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,
}
```

Server Suite Object Reference

```
// 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 objLic in objCollection
for each objLic in objLic
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:

Method	Description
<u>AddLicenseKey</u>	Adds a license key to the set of licenses in the license container.
<u>Commit</u>	Commits any changes to the <code>Licenses</code> object to Active Directory.
<u>GetDirectoryEntry</u>	Returns an instance of the directory entry for the <code>Licenses</code> parent container object.
<u>Refresh</u>	Reloads the <code>Licenses</code> object data from the data in Active Directory.
<u>RemoveLicenseKey</u>	Removes a license key from the set.

Properties

The `Licenses` class provides the following properties:

Property	Description
Count	Gets the number of license objects stored in this <code>L i c e n s e s</code> container.
HasEvaluation	Indicates whether any generated license key installed in the <code>L i c e n s e s</code> parent container is an evaluation license.
HasMachineLicense	Indicates whether any computer license is installed.
ID	Gets the ID from this <code>L i c e n s e s</code> object.
IsReadable	Indicates whether the <code>L i c e n s e s</code> parent object in Active Directory is readable.
IsWritable	Indicates whether the <code>L i c e n s e s</code> parent object in Active Directory is writable.
Item	Gets the <code>L i c e n s e</code> object for a specific type of license.

AddLicenseKey

Adds a license key to the set of licenses in a particular `L i c e n s e s` parent container.

Syntax

```
IKey AddLicenseKey(string key)
```

Parameter

Specify the following parameter when using this method:

Parameter	Description
key	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 `L i c e n s e s` 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 readonly.

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 `L i c e n s e s` object.

Syntax

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

Property value

The ID.

IsReadable

Indicates whether the `L i c e n s e s` parent object in Active Directory is readable for the current user credentials.

Syntax

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

Property value

Returns `true` if the `L i c e n s e s` 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 `L i c e n s e s` object in Active Directory has sufficient permissions to read its properties.

IsWritable

Indicates whether the `L i c e n s e s` parent object in Active Directory is writable for the current user credentials.

Syntax

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

Property value

Returns `true` if the `L i c e n s e s` 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 `L i c e n s e s` 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
type	The type of License object to retrieve from the Licenses 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](../cims/loadlicenses.md)` method.

Methods

The `LicensesCollection` class provides the following methods:

Method	Description
Find	Returns the specific parent container object from the collection of all parent license containers in the forest.
GetEnumerator	Returns an enumeration of <code>Licenses</code> objects.
GetLicensedCount	Returns the total number of licenses of the specified license type that have been installed.
GetUsedCount	Returns the total number of licenses of the specified license type that are currently being used.

Properties

The `LicensesCollection` class provides the following properties:

Property	Description
Count	Gets the number of license container objects stored in the Active Directory forest.
HasEvaluation	Indicates whether there are any evaluation licenses in the forest.
HasMachineLicense	Indicates whether any computer license is installed.
Item	Gets the parent license container object by index number.

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.

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.

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
<code>index</code>	The index number for retrieving the parent license container object

Return value

The parent container object found at the specified index number.

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 Delinea-specific methods and properties for managing individual NIS map records. Only the Delinea-specific methods and properties are described in this reference.

Methods

The Map class provides the following Delinea-specific methods:

Method	Description
Add	Adds a new map entry with the specified key to the NIS map.
Clone	Makes a clone of the NIS map entry. Inherited from <code>ICloneable</code> .
Commit	Commits changes to the NIS map and saves them in Active Directory.

Method	Description
2 Dispose	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from IDisposable.
<u>Exists</u>	Checks whether there is an entry with a specific key-value pair.
<u>Get`](get.md) Returns the NIS map entry with the specified key. </u> <u>[GetByID](getbyid.md) Returns the NIS map entry with the specified ID. </u> <u> [GetDirectoryEntry](getdirectoryentry.md) Returns theDirectoryEntryfor</u> <u>the NIS map object. [GetEnumerator](getenumerator.md) Returns an</u> <u>enumeration ofIMapobjects. [GetRedirectMap](getredirectmap.md) </u> <u>Returns the redirect target NIS map. [Import](import.md) Imports a</u> <u>new map entry with the specified key. [Remove](remove.md) Removes the</u> <u>map entry with the specified key. [RemoveByID</u>	Removes the map entry with the specified ID

Properties

The Map class provides the following Delinea-specific properties:

Property	Description
<u>IsReadable</u>	Indicates whether the map object is readable.
<u>IsWritable</u>	Indicates whether the map object is writable.
<u>Name</u>	Gets or sets the map name of the NIS map.
<u>Store</u>	Gets the Store instance associated with the map object.
<u>Type</u>	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
key	The key you want to check for.
value	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
connection	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
key	The key field that uniquely defines this entry in the NIS map.
value	The value associated with the key field for this entry in the NIS map.
comment	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"
...
```

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\sw0rd"
'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\sw0rd"
'List map names
For each map in objStore
wScript.Echo map.Name
end if
...
```

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\swOrd"
'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.

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
...
```

MzRoleAssignment

Represents a computer-level role assignment.

Syntax

public interface IMzRoleAssignment : IRoleAssignment

Methods

The MzRoleAssignment class provides the following methods:

Method	Description
ClearCustomAttributes	VBScript interface to clear the custom attributes for this class. (Inherited from RoleAssignment .)
Commit	Commits changes in the role assignment to Active Directory. (Inherited from RoleAssignment .)

Method	Description
Delete	Deletes the role. (Inherited from RoleAssignment' (../roleassignment/index.md).) [GetComputer](getcomputer.md) Returns the computer for which the role assignment is made. [GetTrustee](../roleassignment/gettrustee.md) Returns the trustee being assigned. (Inherited from [RoleAssignment] (../roleassignment/index.md).) ICustomAttributeContainer [GetCustomAttributeContainer] (../computerrole/getcustomattributecontainer.md) .NET interface that returns the directory entry for the parent container object for the custom attributes for this class. (Inherited from [RoleAssignment.] (../roleassignment/index.md)) [SetCustomAttribute](../computerrole/setcustomattribute.md) VBScript interface to set the custom attributes for this class. (Inherited from [RoleAssignment] (../roleassignment/index.md).) [ValidateRoleAssignment'] (../roleassignment/index.md).) [ORoleAssignment'] .

Properties

The `MzRoleAssignment` class provides the following properties:

Property	Description
CustomAttributes	VBScript only: Gets or sets custom attributes for this class.
EndTime	Determines the time at which this role becomes inactive. (Inherited from RoleAssignment.)
Id	Gets the GUID of the role assignment. (Inherited from RoleAssignment.)
IsRoleOrphaned	Indicates whether the role assignment is orphaned due to a missing or invalid role. (Inherited from RoleAssignment.)
IsTrusteeOrphaned	Indicates whether the role assignment is orphaned due to a missing or invalid trustee. (Inherited from RoleAssignment.)
LocalTrustee	Gets or sets the local trustee being assigned. (Inherited from RoleAssignment.)
Role	Gets or sets the role the trustee is assigned to. (Inherited from RoleAssignment.)
StartTime	Gets or sets the time from which this role becomes effective. (Inherited from RoleAssignment.)
TrusteeDn	Gets or sets the distinguished name of the trustee assigned this role. (Inherited from RoleAssignment.)
TrusteeType	Gets or sets the trustee type of the role assignment. (Inherited from RoleAssignment.)

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:

Method	Description
Commit	Commits changes in the right to Active Directory. (Inherited from Right .)
Delete	Removes the right. (Inherited from Right .)

Properties

The `NetworkAccess` class provides the following properties:

Property	Description
Description	Gets or sets the description of the right. (Inherited from Right .)
IsReadable	Indicates whether the right is readable. (Inherited from Right .)
IsWritable	Indicates whether the right is writable. (Inherited from Right .)
Name	Gets or sets the name of the right. (Inherited from Right .)
Priority	Gets or sets the priority of this right.

Property	Description
RequirePassword	Gets or sets whether the user's password is required when this right is used.
RunAs	Gets or sets the SID for the run-as user or an SID for the user assigned the right (VBScript).
RunAsList	Gets or sets the SID for the run-as user or a list of SIDs for the users assigned the right (.NET).
RunAsType	Gets or sets the run-as type for the right.
Zone	Gets the zone this right belongs to. (Inherited from Right .)

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

```
ILog\<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:

Method	Description
GetEnumerator	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:

Method	Description
Commit	Commits changes in the right to Active Directory. (Inherited from Right .)
Delete	Deletes the right. (Inherited from Right .)

Properties

The `Pam` class provides the following properties:

Property	Description
Application	Gets or sets the PAM application.
Description	Gets or sets the description of the right. (Inherited from Right .)
IsReadable	Indicates whether the right is readable. (Inherited from Right .)
IsWritable	Indicates whether the right is writable. (Inherited from Right .)
Name	Gets or sets the name of the right. (Inherited from Right .)
Zone	Gets the zone this right belongs to. (Inherited from Right .)

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:

Method	Description
GetEnumerator	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:

Method	Description
Commit	Commits changes in the right to Active Directory.
Delete	Deletes the right.

Properties

The Right class provides the following properties:

Property	Description
Description	Gets or sets the description of the right.
IsReadable	Indicates whether the right is readable.
IsWritable	Indicates whether the right is writable.
Name	Gets or sets the name of the right.
Zone	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:

Method	Description
AddCommand	Adds a command right to the role.
AddNetworkAccess	Adds a network application access right to the role.
AddPamAccess	Adds a PAM application access right to the role.
AddSsh	Adds an SSH application access right to the role.
AddwindowsApplication	Adds a Windows application right to the role.
AddwindowsDesktop	Adds a Windows desktop right to the role.
Assign	Adds a trustee to the role at the zone or computer level.
ClearCustomAttributes	VBScript interface to clear the custom attributes for this class.

Method	Description
Commit	Commits changes to the role to Active Directory.
Delete	Deletes the role from Active Directory.
GetCommands	Returns all command rights added to the role.
ICustomAttributeContainer GetCustomAttributeContainer	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class.
GetNetworkAccesses	Returns the collection of all network application access rights added to this role.
GetPamAccesses	Returns all PAM application access rights added to the role.
GetSshRights	Returns all SSH application access rights added to the role.
GetWindowsApplications	Returns the collection of all Windows application rights added to this role.
GetWindowsDesktops	Returns the collection of all Windows desktop rights added to this role.
IsApplicable	Indicates whether the role is valid in a specified time period.
RemoveAllRights	Removes all rights from the role.
RemoveCommand	Removes a specific command right from the role.
RemoveNetworkAccess	Removes a specific PAM application right from the role.
RemovePamAccess	Removes a specific PAM application right from the role.
RemoveSshRight	Removes a specific SSH application right from the role.
RemovewindowsApplication	Removes a specific Windows application access right from the role.
RemovewindowsDesktop	Removes a specific Windows desktop right from the role.
GetSshRight	Sets a day of the week on which the role is active or inactive.
SetApplicableHour	Sets a day and hour of the week for which the role is active or inactive.
SetCustomAttribute	VBScript interface to set the custom attributes for this class.

Properties

The Role class provides the following properties:

Property	Description
AllowLocalUser	Determines whether the role allows local users.
ApplicableTimeHexString	Gets or sets the time at which the role is active, in hex format.
CustomAttributes	VBScript only: Gets or sets custom attributes for this class.
Description	Gets or sets the description of the role.
Guid	Gets the GUID for this role.
IsReadable	Indicates whether the role is readable.
IsWritable	Indicates whether the role is writable.
Name	Gets or sets the name of the role.
SystemRights	Gets or sets the system rights granted to the role.
Zone	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
command	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 objRole = 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();
...
```

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.

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.

Description

Gets or sets a description of the role.

Syntax

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

Property value

A description of the role.

GetCommands

Returns all the command rights added to this role.

Syntax

```
ICommands 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

```
INetworkAccesses 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.

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`.

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](#).

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.

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
command	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
windowsDesktop	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
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.
isApplicable	Set true to make the role active on the specified day, or false 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 true to make the role active on the specified day and hour, or false 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.

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 Delinea NSS module to be running in “emergency” mode because the adclient 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:

Method	Description
ClearCustomAttributes	VBScript interface to clear the custom attributes for this class.
Commit	Commits changes in the role assignment to Active Directory.
Delete	Deletes the role assignment.
ICustomAttributeContainer GetCustomAttributeContainer	.NET interface that returns the directory entry for the parent container object for the custom attributes for this class.
GetTrustee	Returns the trustee being assigned.
SetCustomAttribute	VBScript interface to set the custom attributes for this class.
Validate	Validates the role assignment

Properties

The RoleAssignment class provides the following properties:

Property	Description
CustomAttributes	VBScript only: Gets or sets custom attributes for this class.
EndTime	Determines the time at which this role assignment becomes inactive.
Id	Gets the GUID of the role assignment.
IsRoleOrphaned	Indicates whether the role assignment is orphaned due to a missing or invalid role.
IsTrusteeOrphaned	Indicates whether the role assignment is orphaned due to a missing or invalid trustee.
LocalTrustee	Gets or sets the local trustee being assigned.
Role	Gets or sets the role the trustee is assigned to.

Property	Description
StartTime	Gets or sets the time from which this role assignment becomes effective.
TrusteeDn	Gets or sets the distinguished name of the trustee assigned this role.
TrusteeType	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.

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.

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.

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`.

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.

RoleAssignments

The `RoleAssignments` class manages a collection of Role Assignment objects.

Syntax

```
public interface IRoleAssignments
```

Methods

The `RoleAssignments` class provides the following method:

Method	Description
GetEnumerator	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:

Method	Description
GetEnumerator	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:

Method	Description
Commit	Commits changes in the right to Active Directory. (Inherited from Right .)
Delete	Deletes the right.(Inherited from Right .)

Properties

The `ssh` class provides the following properties:

Property	Description
Application	Gets or sets the SSH application.
Description	Gets or sets the description of the right. (Inherited from Right .)
IsReadable	Indicates whether the right is readable.(Inherited from Right .)
IsWritable	Indicates whether the right is writable.(Inherited from Right .)
Name	Gets or sets the name of the right.(Inherited from Right .)
Zone	Gets the zone this right belongs to. (Inherited from Right .)

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:

Method	Description
GetEnumerator	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.

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 Delinea-specific methods and properties for managing network or generic NIS maps in a zone. Only Delinea-specific methods and properties are described in this reference.

Methods

The store class provides the following Delinea-specific methods:

Method	Description
Attach	Links the store object to the specified zone.
Clone	Makes a copy of the current Store object instance. Inherited from <code>ICloneable</code> .
Create	Creates a new NIS map object in the zone in Active Directory.
Dispose	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from <code>IDisposable</code> .
Delete	Removes the specified NIS map from the zone and Active Directory.
Exists	Checks whether there is a NIS map with the specified name.
GetDirectoryEntry	Gets the directory entry for the NIS map container.
Open	Opens the NIS map with the specified name.

Properties

The Store class provides the following Delinea-specific properties:

Property	Description
<u>IsReadable</u>	Indicates whether the NIS map store is readable.
<u>IsWritable</u>	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
zonePath	The LDAP path to the zone to which you want to attach the NIS map storage object.
username	The user name to use when linking the NIS map storage object to the specified zone.
password	The user password to use when linking the NIS map storage object to the specified zone.
zoneEntry	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.

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
...
```

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")
...
```

User

The User class enables Delinea 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:

Method	Description
AddUnixProfile	Adds a new UNIX profile for a user to the specified zone.
Commit	Commits the changes to the User object and saves them in Active Directory.
CommitWithoutCheck	Commits the changes to the User object without validating any of the data before saving.
GetDirectoryEntry	Returns an instance of the <code>DirectoryEntry</code> for the user from Active Directory.
GetRoleAssignmentsFromDomain	Returns the collection of all role assignments for a user in a specified domain.
GetRoleAssignmentsFromForest	Returns the collection of all role assignments for a user in a specified forest.
Refresh	Reloads the data in the cache from Active Directory.

Properties

The User class provides the following properties:

Property	Description
AdsIInterface	Gets the ADSI interface of the user object in Active Directory.
ADsPath	Gets the LDAP path to the Active Directory user object.

Property	Description
ID	Gets the UID for the user as a string.
UnixProfiles	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: There is no 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 + ".");
...
```

AdsIInterface

Gets the ADSI interface of the user object in Active Directory.

Syntax

```
IADsUser AdsiInterface {get;}
```

Property value

The ADSI interface of the user object in Active Directory.

Example

The following code sample illustrates using `AdsiInterface` 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 interfce 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 `acme.com`, the LDAP path to the object looks like this:

```
LDAP://cn=john.doe,ou=consultants,dc=acme,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  
...
```

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
domain	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")
...
```

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  
...
```

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  
...
```

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)
```

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:

Method	Description
Commit	Commits any changes to the pending import user object and saves them in Active Directory.
Delete	Marks the pending user profile object for deletion from Active Directory.
Dispose	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from <code>IDisposable</code> .
GetCandidate	Returns the directory object of a user pending import.
Import	Links the pending import user profile with the specified Active Directory user account.
SetCandidate	Sets the directory object of a user pending import.
UpdateStatus	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:

Property	Description
<u>CandidateDN</u>	Gets the distinguished name (DN) of the import candidate.
<u>Gecos</u>	Gets or sets the GECOS field of the UNIX profiles for the pending import user.
<u>HomeDirectory</u>	Gets or sets the home directory for the pending import user.
<u>ID</u>	Gets the unique ID of the pending import user object.
<u>Name</u>	Gets or sets the UNIX user name for a pending import user.
<u>PrimaryGroupID</u>	Gets or sets the UNIX group identifier (GID) of the primary group for the pending import user profile.
<u>Shell</u>	Gets or sets the default login shell for the pending import user.
<u>Source</u>	Gets the text string that describes the source of the pending import data.
<u>Status</u>	Gets the status of the pending import user.
<u>StatusDescription</u>	Gets a text string that provides detailed information about the status of the pending import user.
<u>TimeStamp</u>	Gets the date and time that the pending user profiles were imported from the data source.
<u>UID</u>	Gets or sets the UNIX user identifier (UID) for the pending import user profile.

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.

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. s

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.

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.

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.

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.

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.

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,

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/Acme/Zones/default")  
'Display the time users were 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.

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.

UserInfos

The `UserInfos` class contains methods and properties used to manage a collection of pending import user profiles. This class is defined in the `Centrifury.DirectControl.API.Import` namespace.

Syntax

```
public interface IUserInfos : IDisposable
```

Methods

The `UserInfos` class provides the following methods:

Method	Description
<code>Dispose</code>	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. Inherited from <code>IDisposable</code> .
Find	Returns the pending import user with the specified identifier from the collection of pending user profiles.
GetEnumerator	Returns an enumeration of <code>UserInfo</code> objects.

Properties

The `UserInfos` class provides the following properties:

Property	Description
Count	Gets the total number of pending import user profiles defined in the collection represented by the <code>UserInfos</code> object.
IsEmpty	Indicates whether the collection of pending import users is empty.

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.

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
<code>id</code>	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 `IUserInfo` objects.

Syntax

```
IEnumerator GetEnumerator()
```

Return value

The set of `UserInfo` objects.

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.

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:

Method	Description
Commit	Commits changes to the user profile to Active Directory.

Method	Description
Delete	Marks the user profile for deletion from Active Directory.
GetDirectoryEntry	Returns the directory entry for the UNIX user profile from Active Directory.
GetPrimaryGroup	Returns the UNIX profile of the primary group of the user.
Refresh	Reloads cached object data from Active Directory.
Validate	Checks whether the user profile contains valid data and can be committed to Active Directory.

Properties

The `UserUnixProfile` class provides the following properties:

Property	Description
ADsPath	Gets the LDAP path to the UNIX data object.
Cims	Gets the <code>Cims</code> object for the user.
HomeDirectory	Gets or sets the home directory for the user.
ID	Gets the unique identifier for the <code>UserUnixProfile</code> data object.
IsForeign	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).
IsOrphan	Indicates whether this UNIX user is not associated with a corresponding Active Directory user (not applicable to local user profiles).
IsReadable	Indicates whether the Active Directory object is readable.
IsSFU	Indicates whether this UNIX user is an SFU zone profile (not applicable to local user profiles).
IsWritable	Determines whether the Active Directory object is writable.
Name	Gets or sets the UNIX login name of the user.
PrimaryGroup	Gets or sets the GID of the user's primary group.
ProfileState	Gets or sets the profile state of the local user profile (local user profiles only).
Shell	Gets or sets the default shell for the user.

Property	Description
Type	Gets the <code>UserUnixProfile</code> type for the user.
UnixEnabled	Determines whether the user's UNIX profile is enabled for access to the zone.
User	Gets the user object associated with this user UNIX profile (not applicable to local user profiles).
UserId	Gets or sets the UNIX user identifier (UID).
Zone	Gets the zone object associated with the user.

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.

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
...
```

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.

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
...
```

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 Delinea 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.

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
...
```

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:

Method	Description
GetEnumerator	Returns an enumeration of user profiles.
Refresh	Reloads the collection of UNIX profiles from Active Directory.

Properties

The `UserUnixProfiles` class provides the following properties:

Property	Description
Count	Gets the number of UNIX profiles defined in the collection of <code>UserUnixProfiles</code> .
IsEmpty	Indicates whether the <code>UserUnixProfiles</code> object contains any profiles.

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  
...
```

GetEnumerator

Returns an enumeration of user profiles.

Syntax

```
IEnumerator GetEnumerator()
```

Return value

The set of user profile objects.

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.

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.

WindowsApplication

This class represents a Windows application right.

Syntax

```
public interface IWindowsApplication:IRight
```

Methods

The windowsApplication class provides the following methods:

Method	Description
Commit	Commits changes in the right to Active Directory. (Inherited from Right .)
CreateApplicationCriteria	Creates the match criteria used to identify the Windows application to which you want to grant a right.
Delete	Removes the right. (Inherited from Right .)

Properties

The windowsApplication class provides the following properties:

Property	Description
ApplicationCriteriaList	Gets or sets the properties that are used to identify a specific Windows application.
Description	Gets or sets the description of the right. (Inherited from Right .)
IsReadable	Indicates whether the right is readable. (Inherited from Right .)
IsWritable	Indicates whether the right is writable. (Inherited from Right .)

Property	Description
Name	Gets or sets the name of the right. (Inherited from Right .)
AddLocalGroupPartialProfile	Gets or sets the priority of this right.
RequirePassword	Gets or sets whether the user's password is required when this right is used.
RunAs	Gets or sets the SID for the run-as user or an SID for the user assigned the right (VBScript).
RunAsList	Gets or sets the SID for the run-as user or a list of SIDs for the users assigned the right (.NET).
RunAsType	Gets or sets the run-as type for the right.
Zone	Gets the zone this right belongs to. (Inherited from Right .)

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.

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;
}
```

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.

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

```
IList<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 (../networkaccess/runaslist.md)]` 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.

Method	Description
<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:

Property	Description
<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 partial match.

Property	Description
<u>Description</u>	Gets or sets the description for the match criteria defined for a specific application right.
<u>FileDescription</u>	Gets or sets the file description to match to identify the Windows application associated with this right.
<u>FileDescriptionMatchOption</u>	Specifies whether the string specified for the FileDescription property must be an exact match or a partial match.
<u>FileHash</u>	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.
<u>FileName</u>	Gets or sets the file name to match to identify the Windows application associated with this right.
<u>FileType</u>	Gets or sets the type of executable file to match to identify the Windows application associated with this right.
<u>FileVersion</u>	Gets or sets the file version to match to identify the Windows application associated with this right.
<u>FileVersionMatchOption</u>	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.
<u>IsArgumentCaseSensitive</u>	Specifies whether arguments for the Argument property are case-sensitive.
<u>IsArgumentExactMatch</u>	Specifies whether the string specified for the Argument property must be an exact match or a partial match.
<u>LocalOwner</u>	Gets or sets the local user name or group name to match to allow the use of this Windows application right.
<u>OwnerDN</u>	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.
<u>OwnerSid</u>	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.
<u>OwnerType</u>	Gets or sets the type of owner to match to allow the use of this Windows application right.

Property	Description
<u>Path</u>	Gets or sets the path to the executable to match to identify the Windows application associated with this right.
<u>ProductName</u>	Gets or sets the product name to match to identify the Windows application associated with this right.
<u>ProductNameMatchOption</u>	Specifies whether the string specified for the ProductName property must be an exact match or a partial match.
<u>ProductVersion</u>	Gets or sets the product version to match to identify the Windows application associated with this right.
<u>ProductVersionMatchOption</u>	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.
<u>Publisher</u>	Gets or sets the publisher name to match to identify the Windows application associated with this right.
<u>PublisherMatchOption</u>	Specifies whether the publisher must be an exact match, partial match, start with, or end with the string specified for the Publisher property.
<u>RequireAdministrator</u>	Specifies whether the Windows application associated with this right requires an administrative user account.
<u>SerialNumber</u>	Gets or sets the volume serial number to match to identify the Windows application associated with this right.
<u>SerialNumberMatchOption</u>	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.

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;
    ...
}
```

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
}
```

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 semicolons (;). You can specify the standard system path using the `SYSTEMPATH` keyword, specify a full custom path, such as `C:\Windows\system32\inetrv`, 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
}
```

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.

WindowsApplications

The `WindowsApplications` class manages a collection of Windows application rights.

Syntax

```
public interface IWindowsApplications
```

Methods

The `WindowsApplications` class provides the following method:

Method	Description
GetEnumerator	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	Description
Commit	Commits changes in the right to Active Directory. (Inherited from Right .)
Delete	Removes the right. (Inherited from Right .)

Properties

The WindowsDesktop class provides the following properties:

Property	Description
Description	Gets or sets the description of the right. (Inherited from Right .)
IsReadable	Indicates whether the right is readable. (Inherited from Right .)
IsWritable	Indicates whether the right is writable. (Inherited from Right .)
Name	Gets or sets the name of the right. (Inherited from Right .)
AddLocalGroupPartialProfile	Gets or sets the priority of this right.
RequirePassword	Gets or sets whether the user's password is required when this right is used.
RunAs	Gets or sets the SID for the run-as user or an SID for the users assigned the right (VBScript).
RunAsList	Gets or sets the SID for the run-as user or a list of SIDs for the users assigned the right (.NET).
RunAsType	Gets or sets the run-as type for the right.
Zone	Gets the zone this right belongs to. (Inherited from Right .)

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	Description
GetEnumerator	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:

Method	Description
AddUserRoleAssignment	Adds a role assignment to the user profile.
GetDirectoryEntry	Returns the directory entry for the user from Active Directory.
GetEffectiveUserRoleAssignments	Returns an enumeration of the effective user role assignments.

Properties

The WindowsUser class provides the following property:

Property	Description
<u>Name</u>	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
zone	The zone to which the Windows user belongs.
computer	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 Delinea 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:

Method	Description
GetEnumerator	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 Delinea zone objects (`Centrify.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 Delinea zones interactively using the Access Manager console, see the *Administrator's Guide for Linux and UNIX*.

Methods

The `Zone` class provides the following methods:

Method	Description
<u>AddMitUser</u>	Adds an MIT Kerberos realm-trusted user to this zone.
<u>Commit</u>	Commits settings to Active Directory for the zone object.
<u>CreateImportPendingGroup</u>	Creates a "pending import" group in the zone.
<u>CreateImportPendingUser</u>	Creates a "pending import" user in the zone.
<u>Delete</u>	Deletes the zone object from Active Directory.
<u>GetComputerByDN</u>	Returns the computer profile using the distinguished name (DN) of the profile.
<u>GetComputers</u>	Returns the list of computers in the zone.
<u>GetComputersContainer</u>	Returns the directory entry for the Computers parent container object.
<u>GetDirectoryEntry</u>	Returns the directory entry for the zone.
<u>GetDisplayName</u>	Returns the display name of the zone.
<u>GetGroupsContainer</u>	Returns the directory entry for the Groups parent container object.
<u>GetGroupUnixProfile</u>	Returns the UNIX group profile for a specified group in the zone.
<u>GetGroupUnixProfileByDN</u>	Returns the group profile using the distinguished name (DN) of the profile.
<u>GetGroupUnixProfileByName</u>	Returns the UNIX group profile for a specified group name in the zone.
<u>GetGroupUnixProfiles</u>	Returns the list of UNIX groups in the zone.
<u>GetImportPendingGroup</u>	Returns an individual "pending import" group in the zone.
<u>GetImportPendingGroups</u>	Returns the collection of "pending import" groups in the zone.
<u>GetImportPendingUser</u>	Returns an individual "pending import" user in the zone.
<u>GetImportPendingUsers</u>	Returns the collection of "pending import" users in the zone.
<u>GetLocalGroupsContainer</u>	Returns the DirectoryEntry of the local groups container.
<u>GetLocalGroupUnixProfile</u>	Returns the local UNIX group profile for a specified group name in the zone.

Method	Description
<u>GetLocalGroupUnixProfileByDN</u>	Returns a local group profile using the distinguished name (DN) of the profile.
<u>GetLocalGroupUnixProfileByGid (Int32)</u>	Returns the local group profile using the Group Identifier (GID). This method is exposed to the .COM interface.
<u>GetLocalGroupUnixProfiles</u>	Returns a list of the local group profiles in the zone.
<u>GetLocalUsersContainer</u>	Returns the directory entry of the local users container.
<u>GetLocalUserUnixProfile</u>	Returns the local user profile using the specified user name.
<u>GetLocalUserUnixProfileByDN</u>	Returns the local user profile specified by the distinguished name (DN) of the profile.
<u>GetLocalUserUnixProfileByUid (Int32)</u>	Returns the local user profile using the User Identifier (UID). This method is exposed to the .COM interface
<u>GetLocalUserUnixProfiles</u>	Returns a list of the local user profiles in the zone.
<u>GetUsersContainer</u>	Returns the directory entry for the Users parent container object.
<u>GetUserUnixProfileByDN</u>	Returns the user profile using the distinguished name (DN) of the profile.
<u>GetUserUnixProfileByName</u>	Returns the UNIX user profile for a specified user name in the zone.
<u>GetUserUnixProfiles</u>	Returns the list of UNIX users in the zone.
<u>GroupUnixProfileExists</u>	Indicates whether a UNIX profile exists for the specified group in the zone.
<u>LocalGroupUnixProfileExists</u>	Indicates whether a UNIX profile exists in the zone for the specified local group.
<u>LocalUserUnixProfileExists</u>	Indicates whether a UNIX profile exists in the zone for the specified local user.
<u>PrecreateComputer</u>	Adds a computer to the zone.
<u>PrecreateWindowsComputer</u>	Adds a Windows computer to the zone.
<u>Refresh</u>	Returns the data stored for the zone object from the data in the Active Directory entry.
<u>UserUnixProfileExists</u>	Indicates whether a UNIX profile exists for the specified user in the zone.

Properties

The zone class provides the following properties:

Property	Description
<code>[AdsiInterfaceadsiinterface.md)</code>	Gets the IADs interface of the zone object in Active Directory.
<u>ADsPath</u>	Gets the LDAP path to the zone object.
<u>AgentlessAttribute</u>	Gets or sets the Active Directory attribute used for storing the user's password hash.
<u>AvailableShells</u>	Gets or sets the list of available shells for the zone.
<u>Cims</u>	Gets the Cims object managing the zone.
<u>DefaultGroup</u>	Gets or sets the default group profile to use as the primary group for new users in the zone.
<u>DefaultHomeDirectory</u>	Gets or sets the default path to the user's home directory for new users in the zone.
<u>DefaultShell</u>	Gets or sets the default shell assigned to new users in the zone.
<u>DefaultValueZone</u>	Gets or sets the zone to use for default zone values.
<u>Description</u>	Gets or sets the description property for the zone.
<u>FullName</u>	Gets the full name of the zone.
<u>GroupAutoProvisioningEnabled</u>	Indicates whether auto-provisioning of group profiles is enabled for the zone.
<u>ID</u>	Gets the unique identifier for the zone.
<u>IsHierarchical</u>	Indicates whether this zone supports hierarchical zone features.
<u>IsReadable</u>	Indicates whether the zone object's properties are readable.
<u>IsSFU</u>	Indicates whether the zone uses the Microsoft Services for UNIX (SFU) schema extension.
<u>IsTruncateName</u>	Determines whether the zone is a TruncateName zone.
<u>IsWritable</u>	Indicates whether the zone object's properties are writable.
<u>Licenses</u>	Gets or sets the license container associated with this zone.

Property	Description
<u>MasterDomainController</u>	Gets or sets the name of the primary domain controller for the zone.
<u>MustMaintainADGroupMembership</u>	Determines whether Active Directory group membership must be maintained for UNIX users in the zone.
<u>Name</u>	Gets or sets the name of the zone.
<u>NextAvailableGID</u>	Gets or sets the next available GID value for new groups in the zone.
<u>NextAvailableUID</u>	Gets or sets the next available UID value for new users in the zone.
<u>NextGID</u>	Gets or sets the next GID to be used when adding users.
<u>NextUID</u>	Gets or sets the next UID to be used when adding users.
<u>NISDomain</u>	Gets or sets the NIS domain associated with the zone for SFU zones.
<u>ReservedGID</u>	Gets or sets the list of group identifiers (GIDs) that cannot be assigned in the zone.
<u>ReservedUID</u>	Gets or sets the list of User identifiers (UIDs) that cannot be assigned in the zone.
<u>Schema</u>	Gets the schema type of the zone object.
<u>SFUDomain</u>	Gets or sets the Active Directory domain associated with the zone for SFU zones.
<u>UserAutoProvisioningEnabled</u>	Indicates whether auto-provisioning of user profiles is enabled for the zone.
<u>Version</u>	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.
<code>shell</code>	The default shell for the user in the specified zone.
<code>homeDir</code>	The default login shell for the user in the specified zone.
<code>primaryGroup</code>	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
...
```


AdsInterface

Gets the IADs interface of the zone object from Active Directory.

Syntax

```
IADs AdsInterface {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/Acme/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.

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
...
```

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/Acme/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/Acme/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
...
```

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/Acme/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
...
```

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
...
```

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=centrify,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=centrify,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=arcade,
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. [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=arcade,
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
group	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=arcade,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=arcade,
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
id	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" on page ccxviii 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
<code>dn</code>	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 (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.

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/Acme/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
...
```

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*.

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
...
```

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.

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
userName	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.

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` throw 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/Acme/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 (`NextAvailableGID`] (`nextavailablegid.md`)) 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
...
```

PrecreateComputer

Adds a computer to a zone.

Syntax

```
IComputer PrecreateComputer(DirectoryEntry adComputerEntry, string[] spn, DirectoryEntry trustee)
```

```
IComputer PrecreateComputer(DirectoryEntry containerEntry, string cn, string dnsName, string [] spn, DirectoryEntry trustee
```

```
IComputer PrecreateComputer(DirectoryEntry adComputerEntry, string[] spn, DirectoryEntry trustee, bool skipPermissionSetting);
```

```
IComputer PrecreateComputer(DirectoryEntry containerEntry, string cn, string dnsName, string [] spn, DirectoryEntry trustee, bool skipPermissionSetting);
```

Parameters

Specify the following parameters when using this method.

Parameter	Description
adComputerEntry	The DNS host name of the Active Directory computer object you wish to add to the zone.
containerEntry	The Directory container for the created computer.
cn	The computer name.
dnsName	The DNS name of the created computer.
skipPermissionSetting	Specifies if permission delegation is skipped when precreating computers.
spn	Service Principal Name. Specify null to use default.
trustee	The user or group to delegate adjoint permissions to, Specify null to delegate the permission for a self-service join.
trusteeDn	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 `PrecreateCompute(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);  
IComputer PrecreateWindowsComputer(DirectoryEntry adComputerEntry, bool  
skipPermissionSetting);
```

Parameters

Specify the following parameter when using this method.

Parameter	Description
adComputerEntry	The DNS host name of the Windows computer object you wish to add to the zone.
skipPermissionSetting	Specifies if permission delegation is skipped when precreating computers.

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  
...
```

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 Delinea zones
- Standard Delinea RFC 2307-compliant zones
- Delinea 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 Centify zones are:

Schema name	Value	Description
Unknown	-1	Schema unknown
Dynamic_Schema_1_0	0	Standard Delinea zone, version 1.x Uses the Delinea version 1.x and standard Active Directory schema data storage model. This zone type is for backward compatibility and otherwise no longer in use.

Schema name	Value	Description
Dynamic_Schema_2_0	1	Standard Delinea zone, version 2.x and 3.x Uses the Delinea 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 Delinea 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.
SFU_4_0	3	SFU zone, version 4.x Uses a combination of the Delinea 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 Delinea zone, version 3.x and 4.x Uses the Delinea version 4.x and Active Directory schema data storage model. Note: The only difference between the Dynamic_Schema_2_0 data storage model and the Dynamic_Schema_3_0 data storage model is the use of the managedBy attribute. This attribute is set in zones that use the Dynamic_Schema_2_0 schema. The managedBy attribute is not used in zones that use in the Dynamic_Schema_3_0 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. Note: The only difference between the CDC_RFC_2307 data storage model and the CDC_RFC_2307_2 data storage model is the use of the managedBy attribute. This attribute is set in zones that use the CDC_RFC_2307 schema. The managedBy attribute is not used in zones that use in the CDC_RFC_2307_2 schema.

Schema name	Value	Description
Dynamic_Schema_5_0	8	Hierarchical zone, version 5.x Uses the Delinea version 5.x and standard Active Directory schema data storage model. Note: The difference between the Dynamic_Schema_5_0 data storage model and the CDC_RFC_2307_3 data storage model is that in the Dynamic_Schema_5_0 storage model, all Delinea data is stored as part of the zone. In the CDC_RFC_2307_3 storage model, user and group attributes are stored as part of the User and Group 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*.

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
...
```

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
...
```

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 section explains the following formats:

- [Hex string](#)
- [Hour mapping](#)
- [Day mapping](#)

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:

Reading and Setting Timebox Values

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.

Reading and Setting Timebox Values

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)