

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

*ADEdit Command Reference and Scripting Guide*

August 2019 (release 19.6)

Centrify Corporation





## Legal Notice

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

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

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

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

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

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

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

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



# Contents

<b>About this guide</b> .....	<b>13</b>
Intended audience .....	13
Using this guide .....	13
Viewing command help .....	14
Documentation conventions .....	15
Finding more information about Centrify products .....	15
Product names .....	16
Contacting Centrify .....	18
Getting additional support .....	18
<b>Introduction</b> .....	<b>19</b>
How ADEdit uses Tcl .....	19
What ADEdit provides .....	19
How ADEdit works with other Centrify components .....	22
ADEdit components .....	24
ADEdit context .....	25
Logical organization for ADEdit commands .....	27
<b>Getting started with ADEdit</b> .....	<b>29</b>
Starting ADEdit for the first time .....	29
Basic command syntax .....	29
Learning to use ADEdit .....	32
Binding to a domain and domain controller .....	34
Selecting an object .....	37
Creating a new object .....	39
Examining objects and context .....	39
Modifying or deleting selected objects .....	41



Saving selected objects .....	42
Pushing and popping context .....	42
Creating ADEdit scripts .....	42
<b>ADEdit commands organized by type .....</b>	<b>47</b>
General-purpose commands .....	47
Context commands .....	47
Object-management commands .....	48
Utility commands .....	57
Security descriptor commands .....	58
<b>Using the demonstration scripts .....</b>	<b>59</b>
Zone containers and nodes .....	60
Create Tcl procedures .....	62
Reading command line input .....	64
Create a parent zone .....	66
Create child zones .....	68
Create privileged commands and roles .....	70
Add and provision UNIX users .....	75
Simple tools .....	78
Run a script from a script .....	84
<b>ADEdit command reference .....</b>	<b>90</b>
add_command_to_role .....	91
add_map_entry .....	93
add_map_entry_with_comment .....	95
add_object_value .....	97
add_pamapp_to_role .....	99



add_sd_ace .....	101
bind .....	105
clear_rs_env_from_role .....	108
create_computer_role .....	110
create_zone .....	113
delegate_zone_right .....	117
delete_dz_command .....	121
delete_local_group_profile .....	123
delete_local_user_profile .....	126
delete_map_entry .....	128
delete_nis_map .....	130
delete_object .....	132
delete_pam_app .....	134
delete_role .....	136
delete_role_assignment .....	139
delete_rs_command .....	140
delete_rs_env .....	142
delete_sub_tree .....	144
delete_zone .....	147
delete_zone_computer .....	149
delete_zone_group .....	151
delete_zone_user .....	153
dn_from_domain .....	154
dn_to_principal .....	156
domain_from_dn .....	158
explain_sd .....	159
forest_from_domain .....	163
get_adinfo .....	164



get_bind_info .....	166
get_child_zones .....	169
get_dz_commands .....	171
get_dzc_field .....	173
get_group_members .....	180
get_local_group_profile_field .....	182
get_local_groups_profile .....	185
get_local_user_profile_field .....	188
get_local_users_profile .....	192
get_nis_map .....	194
get_nis_map_field .....	196
get_nis_map_with_comment .....	199
get_nis_maps .....	201
get_object_field .....	203
get_object_field_names .....	205
get_objects .....	208
get_pam_apps .....	211
get_pam_field .....	213
get_parent_dn .....	216
get_pending_zone_groups .....	217
get_pending_zone_users .....	219
get_pwnam .....	221
get_rdn .....	223
get_role_apps .....	224
get_role_assignment_field .....	227
get_role_assignments .....	229
get_role_commands .....	232
get_role_field .....	234



get_role_rs_commands .....	238
get_role_rs_env .....	240
get_roles .....	242
get_rs_commands .....	244
get_rs_envs .....	246
get_rsc_field .....	248
get_rse_cmds .....	252
get_rse_field .....	254
get_schema_guid .....	256
get_zone_computer_field .....	257
get_zone_computers .....	260
get_zone_field .....	262
get_zone_group_field .....	267
get_zone_groups .....	270
get_zone_nss_vars .....	272
get_zone_user_field .....	274
get_zone_users .....	277
get_zones .....	279
getent_passwd .....	281
guid_to_id .....	283
help .....	285
is_dz_enabled .....	286
joined_get_user_membership .....	288
joined_name_to_principal .....	290
joined_user_in_group .....	292
list_dz_commands .....	293
list_local_groups_profile .....	296
list_local_users_profile .....	298



list_nis_map .....	301
list_nis_map_with_comment .....	303
list_nis_maps .....	305
list_pam_apps .....	308
list_pending_zone_groups .....	310
list_pending_zone_users .....	312
list_role_assignments .....	314
list_role_rights .....	317
list_roles .....	319
list_rs_commands .....	321
list_rs_envs .....	324
list_zone_computers .....	326
list_zone_groups .....	328
list_zone_users .....	330
manage_dz .....	333
move_object .....	335
new_dz_command .....	336
new_local_group_profile .....	338
new_local_user_profile .....	341
new_nis_map .....	345
new_object .....	347
new_pam_app .....	350
new_role .....	352
new_role_assignment .....	354
new_rs_command .....	357
new_rs_env .....	359
new_zone_computer .....	361
new_zone_group .....	363



new_zone_user .....	366
pop .....	368
principal_from_sid .....	370
principal_to_dn .....	371
principal_to_id .....	373
push .....	375
quit .....	376
remove_command_from_role .....	378
remove_object_value .....	380
remove_pamapp_from_role .....	383
remove_sd_ace .....	385
rename_object .....	389
save_dz_command .....	390
save_local_group_profile .....	392
save_local_user_profile .....	395
save_nis_map .....	398
save_object .....	400
save_pam_app .....	402
save_role .....	404
save_role_assignment .....	406
save_rs_command .....	408
save_rs_env .....	409
save_zone .....	411
save_zone_computer .....	413
save_zone_group .....	415
save_zone_user .....	417
select_dz_command .....	419
select_local_group_profile .....	421



select_local_user_profile .....	424
select_nis_map .....	427
select_object .....	429
select_pam_app .....	431
select_role .....	434
select_role_assignment .....	436
select_rs_command .....	439
select_rs_env .....	441
select_zone .....	443
select_zone_computer .....	446
select_zone_group .....	448
select_zone_user .....	451
set_dzc_field .....	453
set_ldap_timeout .....	460
set_local_group_profile_field .....	461
set_local_user_profile_field .....	465
set_object_field .....	469
set_pam_field .....	472
set_role_assignment_field .....	475
set_role_field .....	477
set_rs_env_for_role .....	482
set_rsc_field .....	484
set_rse_field .....	490
set_sd_owner .....	492
set_user_password .....	495
set_zone_computer_field .....	497
set_zone_field .....	499
set_zone_group_field .....	503



set_zone_user_field .....	506
show .....	509
sid_to_escaped_string .....	511
sid_to_uid .....	513
validate_license .....	515

## ADEdit Tcl procedure library reference ..... 517

add_user_to_group .....	517
convert_msdate .....	519
create_adgroup .....	520
create_aduser .....	522
create_assignment .....	523
create_dz_command .....	525
create_group .....	527
create_nismap .....	529
create_pam_app .....	530
create_role .....	532
create_rs_command .....	533
create_rs_env .....	535
create_user .....	537
decode_timebox .....	539
encode_timebox .....	541
explain_groupType .....	542
explain_ptype .....	544
explain_trustAttributes .....	545
explain_trustDirection .....	547
explain_userAccountControl .....	548
get_all_zone_users .....	549



get_effective_groups .....	551
get_effective_users .....	552
get_user_groups .....	554
get_user_role_assignments .....	556
list_zones .....	558
lmerge .....	559
modify_timebox .....	561
precreate_computer .....	563
remove_user_from_group .....	567
set_change_pwd_allowed .....	569
set_change_pwd_denied .....	570
<b>Timebox value format .....</b>	<b>572</b>
Hex string .....	572
Hour mapping .....	572
Day mapping .....	574
<b>Using ADEdit with classic zones .....</b>	<b>576</b>
Enabling authorization in classic zones .....	576
Working with privileged commands and PAM applications .....	577
Working with restricted shell environments and commands .....	577
Creating computer-level role assignments in classic zones .....	579
<b>Quick reference for commands and library procedures .....</b>	<b>581</b>



# About this guide

This *AEdit Command Reference and Scripting Guide* describes how to use the Centrify AEdit command-line interface to manage Centrify objects stored in Microsoft Active Directory. AEdit is a Tool command language (Tcl) application that enables administrators to run commands and write scripts that modify data in Active Directory directly from their Linux or UNIX console.

## Intended audience

This guide describes AEdit for UNIX administrators who want to manage Centrify and Active Directory from a Linux, UNIX, or Mac computer through CLI commands or scripts. It assumes that you are well-versed in Active Directory's architecture and management, and that you're equally well-versed in Centrify access control and privilege management features. For more complete information about Centrify software and management tasks, see the *Administrator's Guide for Linux and UNIX*.

## Using this guide

This guide describes how to use AEdit and provide reference information for all AEdit commands and the AEdit library. It does not describe how to write Tcl scripts using AEdit commands. For a comprehensive explanation of Tcl and its use, see *Tcl and the Tk Toolkit* by John K. Ousterhout and Ken Jones (published by Addison-Wesley).

The chapters provide the following information:

- [Introduction](#) describes the basic features of AEdit and the types of commands it offers, including how it fits in with other components of Centrify software.



- [Getting started with ADEdit](#) describes the basics of ADEdit command syntax and the logical flow of commands that you need to be familiar with before you begin executing interactive ADEdit sessions or writing ADEdit scripts.
- [ADEdit commands organized by type](#) assembles the ADEdit commands into logical groups, corresponding to their usage, and summarizes each command.
- [Using the demonstration scripts](#) provides script samples for a series of common tasks that you can incorporate into your scripts.
- [ADEdit command reference](#) provides full command descriptions in alphabetical order.
- [ADEdit Tcl procedure library reference](#) describes the Tcl procedures available in the `ade_lib` Tcl library that use ADEdit commands to perform common administrative tasks.
- [Timebox value format](#) describes the format of the timebox value used to set hours of the week when a role is enabled and disabled.
- [Using ADEdit with classic zones](#) summarizes the differences between working with classic and hierarchical zone and lists the commands that are specifically for managing authorization in classic zones.
- [Quick reference for commands and library procedures](#) provides a summary of all ADEdit commands and procedures, including the command syntax and abbreviations.

## Viewing command help

ADEdit provides brief help text for each command. To view the help, enter `help command_name` from the ADEdit command prompt. For example, to see the help for the `validate_license` command you would enter the following:

```
>help validate_license
```

You can also display the general help text for ADEdit by entering `man adedit` from the shell.



## Documentation conventions

The following conventions are used in Centrifly documentation:

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

## Finding more information about Centrifly products

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

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

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



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

## Product names

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

Current Overall Product Name	Current Services Available
Centriify Zero Trust Privilege Services	Privileged Access Service
	Gateway Session Audit and Monitoring
	Authentication Service
	Privilege Elevation Service
	Audit and Monitoring Service
	Privilege Threat Analytics Service

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

Previous Product Offering	Previous Product Offering	Description	Current Product Offering
	Centriify Privileged Service (CPS)		Privileged Access Service
DirectControl (DC)			Authentication Service
DirectAuthorize (DZ or DZwin)			Privilege Elevation Service
DirectAudit (DA)			Audit and Monitoring Service
	Infrastructure Services		Privileged Access Service, Authentication Service, Privilege Elevation Service, Audit and Monitoring Service, and Privilege



Previous Product Offering	Previous Product Offering	Description	Current Product Offering
Threat Analytics Service			
DirectManage (DM)	Management Services	Consoles that are used by all 3 services: Authentication Service, Privilege Elevation Service, and Audit and Monitoring Service	
DirectSecure (DS)	Isolation and Encryption Service		Still supported but no longer being developed or updated
	User Analytics Service		Privilege Threat Analytics Service

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

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



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

## Contacting Centrify

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

## Getting additional support

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

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

# Introduction

Centrify ADEdit is a command-line interface (CLI) utility that enables UNIX administrators to manage Centrify objects—such as zones, rights, and roles—in Microsoft Active Directory. This chapter introduces you to ADEdit’s main features and architecture.

## How ADEdit uses Tcl

ADEdit is implemented as a Tcl application. Tcl (Tool Command Language) is a powerful but easy to learn programming language that provides full scripting ability. With Tcl, administrators can write simple management scripts that perform complex tasks with a single execution. Experienced Tcl programmers can also include ADEdit commands in their own Tcl applications to add Centrify management capabilities and GUI interfaces for ADEdit operations to those applications.

Administrators who aren’t familiar with Tcl can use ADEdit as a scripting tool on their Linux or UNIX computer to manage Centrify directly from the command line or by combining commands into scripts.

## What ADEdit provides

The purpose of ADEdit is to let an administrator with the proper Active Directory permissions fully manage Centrify objects from a UNIX console. By using ADEdit, for example, an administrator working on a Linux computer can perform common administrative tasks such as create a new user account, add a user to a new group, or assign a user to a new role. That same administrator might also query Active Directory for information about zones, groups, roles, or any other Centrify objects.



Because ADEdit is a more powerful and flexible tool, it is intended to replace some of Centrify's previous-generation UNIX command line programs such as `adupdate` and `adquery`. Those previous-generation tools limited the operations administrators could perform to a computer's currently joined zone and domain. With ADEdit, administrators can manage objects in any zone or domain and perform operations on many more features than were possible using its predecessors.

To give administrators additional flexibility for performing administrative tasks, ADEdit also allows for multiple modes of execution and provides its own accompanying library of predefined scripts for common tasks.

## Administration across domains and forests

ADEdit offers complete control of Centrify objects and properties from a Linux or UNIX console. Administrators with the proper permissions on the Active Directory domain controller can modify every aspect of operation that the Access Manager offers. For example, administrators can use ADEdit to create zones, add groups, delegate permissions, define roles, and modify user properties, group membership and role assignments.

ADEdit can operate on any domain in any forest. Its host computer does not need to be joined to a domain to work with that domain. As long as the administrator has the necessary authentication and rights to work on a domain, ADEdit can bind to the domain and work on it. ADEdit can also work simultaneously on multiple domains in multiple forests.

ADEdit enables you to manage all aspects of the access control and privilege management features of multiple Centrify software from a single CLI tool. For example, it can replace `adupdate` and `adquery` and offers the features of LDAP clients such as `1dapsearch`, without the limitations of those command line programs.

## Options for execution

ADEdit offers multiple modes of execution:

- **Interactive mode.** In interactive mode, ADEdit executes single CLI commands in real time. You can enter a series of commands within a



shell to perform simple administrative tasks. ADEdit offers command history that is persistent from session to session. You can use the up arrow and Enter keys to review and re-enter commands instead of retyping complete commands from scratch.

- **Script execution.** ADEdit can accept and execute a Tcl script file that includes ADEdit commands. The Tcl scripting language includes full programming logic with variables, logical operators, branching, functions (called *procedures* in Tcl), and other useful program-flow features. As the script executes, ADEdit keeps the Active Directory objects that it is working on in internal memory. It does not require repeated queries to Active Directory as it works on an object.
- **Executable file.** You can set up any ADEdit Tcl script as an executable file that can run by itself on a UNIX platform.

Scripting makes ADEdit a very flexible administration tool. You can use a single script to handle hundreds or thousands of repetitive tasks that would take a very long time to perform through the console. And you can write a set of scripts to quickly and easily check on and respond to current conditions. A script could, for example, create a new zone, read `etc/passwd` files on UNIX computers in that zone, and migrate all existing UNIX users it finds there into new zone user accounts. Another script could find users in specified groups and then assign a new role to all users in those groups.

With that power comes responsibility. It's quite possible for an ADEdit script—or even a single ADEdit command—to completely erase Active Directory's contents if used incorrectly. There are, for the most part, no warnings and there is no undo feature if this happens. Only knowledgeable users should use ADEdit, and it is important to test scripts in sample environments before deploying them to the enterprise.

## Library of predefined procedures

ADEdit installs with an accompanying library of utility procedures called the `ade_lib` Tcl library. These procedures use ADEdit commands to perform standard administrative operations such as adding zone users to a zone group or creating a new Active Directory user. The procedures in the library also provide examples of how to use ADEdit commands efficiently in Tcl scripts. From these examples, administrators can learn how to use and adapt ADEdit commands in their own custom scripts.



## How ADEdit works with other Centrify components

ADEdit is part Centrify Authentication Service, Privilege Elevation Service, and Audit and Monitoring Service and works with specific Windows and UNIX components of the Centrify architecture. As described in the *Administrator's Guide for Linux and UNIX*, Centrify uses Active Directory, which runs in a Windows network, to store Centrify-specific data such as zone information. To make computers part of an Active Directory domain, administrators deploy a platform-specific Centrify agent. After the agent is deployed and the computer joins an Active Directory domain, the computer is a Centrify-managed computer and ADEdit can define, retrieve, modify, and delete Active Directory and Centrify information for that computer.

### Active Directory and ADEdit

Active Directory uses multi-master data storage. It replicates directory data on multiple domain controllers throughout a domain. Changes in data on one domain controller are replicated to the other domain controllers in the domain.

To perform virtually any operation, ADEdit must bind to one or more Active Directory domain controllers. ADEdit can then query Active Directory for data within bound domains, retrieve Active Directory objects, modify retrieved objects, create new objects, and delete existing objects. Those objects include all Centrify-specific objects such as zone objects, zone user objects, role objects, and more.

**Note** ADEdit is not limited in scope to Centrify-specific information. An administrator with full privileges could define, retrieve, modify, and delete information for any object or attribute in Active Directory.

### Managed computers and ADEdit

For computers to be managed by Centrify, they must have the Centrify agent installed and must be joined to an Active-Directory domain. The Centrify agent includes the following components that work directly with ADEdit:



- **adclient** is a Centrify process running on a managed computer. The `adclient` process communicates with Active Directory to make its host computer part of the Active Directory domain. Applications that require authentication and authorization or other services then use `adclient` to query Active Directory for that information.

In most cases, ADEdit connects directly to Active Directory without using `adclient`. However, there are some commands that use `adclient` to get information more efficiently than from Active Directory directly.

- **Centrify command line programs** are commands administrators can run on managed computers to control `adclient` operations and work with the Centrify data stored in Active Directory. ADEdit replaces some of these commands, but occasionally works in conjunction with other commands such as `adflush`, especially when executing ADEdit commands that work through `adclient`. For more information about using command line programs, see the *Administrator's Guide for Linux and UNIX*.

## Other administrative options

ADEdit is intended to be the primary tool for administrators who want to perform administrative tasks directly from a command line or in scripts on Linux, UNIX, and Mac OS X computers. However, there are two other administrative options for performing the same tasks outside of ADEdit:

- The **Access Manager** console runs on a Windows computer and provides a graphical user interface that you can use for complete control of Centrify-related information and some Active Directory features.
- The **Centrify Authentication Service, Privilege Elevation Service, and Audit and Monitoring Service SDK for Windows** provides application programming interfaces (API) that you can use to control all of the same features provided the **Access Manager** console.

It's important to realize when using any of these tools that an instance of one of these tools has no knowledge of other tool instances and acts as if it's the only administrative tool at work. For example, if one administrator uses the Access Manager console to modify a zone object at the same time as another administrator uses ADEdit to modify the same zone object, their changes might clash. For example, if the changes are first saved by the administrative



Access Manager, those changes might be overridden by changes saved by ADEdit. The last tool to save object data has the final say.

This is true as well for different instances of ADEdit. If two administrators both use different ADEdit instances simultaneously to work on the same object, the administrator who last saves the object is the only one whose work will have an effect on the object.

It's important when using ADEdit in an environment with multiple administrators to retrieve an object, make changes, and check it back in efficiently to avoid conflicts. ADEdit object changes are not atomic.

It helps to bind all administration tools to the same domain controller within a domain to further minimize conflicts. If tools work on different domain controllers, one tool's changes may take time to replicate to the other domain controllers, so other tools connected to other domain controllers won't be able to see those changes immediately.

## ADEdit components

ADEdit has two components: the ADEdit application and the `ade_lib` Tcl library. They are both installed when the Centrify agent is installed on a Linux, UNIX, or Mac OS X computer to be managed.

A user can access ADEdit through a CLI in a shell or through an executing Tcl script or Tcl application. ADEdit's Tcl interpreter executes the commands it receives from the CLI using the ADEdit commands and Tcl commands that are part of ADEdit. It may also use `ade_lib` Tcl library commands if specified. Tcl scripts and applications use ADEdit's commands and `ade_lib` Tcl library commands directly. ADEdit binds to an Active Directory domain controller, with which it exchanges data. ADEdit may also (in a few cases) get data from Active Directory through the `adcClient` process.

### The ADEdit application

ADEdit uses Tcl as its scripting language. Tcl is a long-established extensible scripting language that offers standard programming features and an extension named Tk that creates GUIs simply and quickly. Tcl is described in



the authoritative book *Tcl and the Tk Toolkit* by John K. Ousterhout and Ken Jones (Addison-Wesley, 2010).

ADEdit includes a Tcl interpreter and the Tcl core commands, which allow it to execute standard Tcl scripts. ADEdit also includes a set of its own commands designed to manage Centrify and Active Directory information.

ADEdit will execute individual commands in a CLI (in interactive mode) or sets of commands as an ADEdit script.

## The `ade_lib` Tcl library

The `ade_lib` Tcl library is a collection of Tcl procedures that provide helper functions for common Centrify-related management tasks such as listing zone information for a domain or creating an Active Directory user. You can include `ade_lib` in other ADEdit scripts to use its commands.

To use `ade_lib` in a Tcl script or in an ADEdit session, begin the script or session with:

```
package require ade_lib
```

## ADEdit context

When ADEdit commands work on Active Directory objects, they don't specify a domain and the object to work on as part of each command. ADEdit instead maintains a context in memory that defines what commands work on.

ADEdit's context has two types of components:

- **A set of one or more bindings that connect ADEdit to domains in the forest.** Each binding uses an authentication to connect to an Active Directory domain controller. The authentication must have enough rights to perform ADEdit's administrative actions on the domain controller. Each binding binds ADEdit to a single domain; multiple bindings bind ADEdit to multiple domains at one time.
- **A set of zero, one, or more selected Active Directory objects that ADEdit works on.** A selected object is typically a Centrify object such as a zone, zone user, role, or NIS map, but can also be any generic Active Directory object. ADEdit stores each selected object with all of its



attributes (called fields within ADEdit). ADEdit stores no more than one type of each selected object: one zone object, for example, one PAM application object, one generic Active Directory object, and so on.

An ADEdit session or script typically starts by binding to one or more domains. If ADEdit isn't bound to a domain, none of its commands that work with Active Directory (which is most of them) have any effect. Once bound, ADEdit commands work within the scope of all currently bound domains.

An ADEdit session or script then typically selects an object to work on: it specifies an object such as a zone user object that ADEdit retrieves from Active Directory and stores in memory as part of the context. All subsequent zone user commands then work on the zone user object in memory, not the zone user object as it is stored in Active Directory.

When finished with a selected object, the session or script can simply ignore the object (if nothing has changed in it) or it can save the object back to Active Directory (if the object has been modified and modifications need to go back to Active Directory, overwriting the object there). The selected object remains stored in ADEdit's context until the session or script selects a new object of the same type, which replaces the previous object.

By maintaining a context with selected objects, ADEdit avoids constant Active Directory queries for successive object management commands: A selection command queries Active Directory to retrieve an object. Reading or modifying object fields occurs internally and doesn't require Active Directory queries. If the object is saved, a final Active Directory query returns the modified object to Active Directory.

## Context persistence

ADEdit's context persists for the duration of an ADEdit interactive session. The context in an ADEdit script persists only until the end of the script's execution.

## Pushing and popping contexts

ADEdit can save and retrieve contexts using push and pop commands that use a stack to store successive levels of context. Pushing and popping contexts is useful within Tcl scripts when jumping to a procedure. The script can push the



current context to the stack, create an entirely new context for the procedure, then pop the original context back when exiting the procedure.

## Context cautions

Working with ADEdit's context requires some thought. Commands that affect objects don't explicitly specify an object, so you must be careful to ensure that the correct object is specified before executing commands that affect the object. ADEdit has context reporting commands that help by showing current domain bindings and selected objects.

It's important to realize that any modifications to a selected object have no effect until the object is saved back to Active Directory. If you forget to save an object, you lose all modifications.

If you keep an object in context a long time between selecting the object and saving the object, be aware—as noted earlier—that another administration tool may alter the object in Active Directory during that time and you won't know about those alterations.

## Logical organization for ADEdit commands

The commands you can execute with ADEdit fall into the following logical categories:

- **General-purpose commands** that control ADEdit operation and provide information about ADEdit.  
For example, you use these commands to view usage help, set the LDAP query time-out interval, and quit ADEdit.
- **Context commands** that set up and control the ADEdit domain context.  
For example, you use these commands to bind to a domain before subsequent object management commands, view current bindings, and change the context.
- **Object management commands** that enable you to perform all of the same tasks as you can with Active Directory Users and Computers and Access Manager.



For example, you use these commands to create, select, and manage zones, users, groups, computers, rights, roles and role Assignments.

- **Utility commands** that perform useful data retrieval and data conversion tasks.

For example, you use these commands to convert domain names and security principal names from one format to another.

- **Security descriptor commands** that modify security descriptors and make them readable.

For example, you use these commands to convert security descriptors strings from one format to another.

For more information about the commands each category, see [ADEdit commands organized by type](#). For details about specific commands, see [ADEdit command reference](#).

# Getting started with ADEdit

This chapter describes ADEdit's basic syntax, shows the typical logic flow used to handle Centrify objects, and describes in detail the steps in that logic flow using simple examples.

## Starting ADEdit for the first time

The ADEdit application (`adedit`) and accompanying library of Tcl procedures (`ade_lib`) are installed automatically when you install the Centrify agent on a UNIX, Linux, or Mac OS X computer. Therefore, both the application and the library are immediately available on any Centrify-managed computer. You are not required to join the domain before using ADEdit for the first time.

To start a new interactive ADEdit session, type `adedit` in a standard shell after logging on to your computer. A new angle bracket (`>`) prompt indicates that you are in an interactive ADEdit session. For example:

```
[myprompt]$ adedit  
>
```

Anyone can launch ADEdit. However, only users who have sufficient privileges can modify Active Directory objects and Centrify-specific data.

## Basic command syntax

ADEdit includes a Tcl interpreter and uses Tcl syntax. However, ADEdit commands have their own syntax within the Tcl syntax. Like other Tcl commands, ADEdit commands are always completely lowercase. ADEdit does not recognize commands with uppercase characters.



## Arguments and options

An ADEdit command works very much like a UNIX command. Depending on the command, you might be required to specify one or more arguments. An argument is typically a variable that follows the command name to provide data that controls the operation to be performed. In some cases, values for the variables are required for a command to execute. In other cases, variables might be optional. The reference information for individual commands indicates whether arguments are required or optional. In most cases, however, arguments must be entered in the order specified for the command.

In addition to arguments, ADEdit commands may or may not have options. Options must precede a command's arguments. Each option is a single word preceded by a hyphen (-) such as `-write`. Options can also have their own arguments. If an option takes an argument, it must immediately follow the option.

Options are used to control specific operations of ADEdit commands. For example:

```
>bind -gc acme.com administrator #3gEgh^&4
```

In this example, the `bind` command has an option `-gc` that specifies a global catalog domain controller. Three arguments follow the option. The first argument is required and specifies the domain to which to bind. The second and third arguments are optional and provide a use name and password to be used for binding.

## Command execution and results

Like most UNIX commands, ADEdit produces no output or return value if a command executes successfully. Only commands that are defined to return a result produce output when an operation completes successfully. If a command fails, however, ADEdit notifies you of an error in execution and reports the general reason for failure. For example, you might see an error message indicating the wrong number of arguments or a connection problem.

Some commands return results as a Tcl list that other commands in a Tcl script can use. Other commands output results directly to standard output



`stdout`) where the results are displayed in the shell. You can redirect a command's `stdout` output to a file or other destination, if desired.

Commands that return Tcl lists start with `get` followed by an object type (`get_zone_users`, for example) and return the list of the objects matching the specified object type that are stored in Active Directory. Because other commands can use the Tcl list to act on the returned data, the `get` commands are especially useful for writing scripts.

Commands that send data to `stdout` start with `list` followed by an object type (`list_zone_groups`, for example) and return the list of the objects matching the specified object type that are stored in Active Directory for the currently selected context. Because the list goes to `stdout`, the `list` commands are especially useful for displaying data in interactive sessions as a script executes.

## Using command abbreviations

Most ADEdit commands have an abbreviation that you can use in place of the full command name. For example, the command `list_zone_users` has the abbreviation `lszu`. You can use either the full command name or the abbreviation for any command.

## Using the command history

ADEdit in an interactive session retains a history of previously entered commands. You can visit the command history by pressing the up arrow key to go back in the history and the down arrow key to go forward. Press Enter to run the current command.

ADEdit retains its command history across sessions, so if you quit ADEdit and restart it, you can still visit commands entered in the previous session. The command history has a 50-command capacity. Once full, the history drops old commands as you enter new commands.



## Using the help command

The ADEdit `help` command provides brief information about ADEdit commands. If you enter `help` in ADEdit followed by a command or command abbreviation, `help` returns information about that command, including its syntax.

You can use the wildcard character `*` to specifying any number of variable characters or `?` to specify a single variable character within a command string following the `help` command. The `help` command returns help text for all commands that match the wildcard string. For example, the following command returns help for all commands that start with `get`.

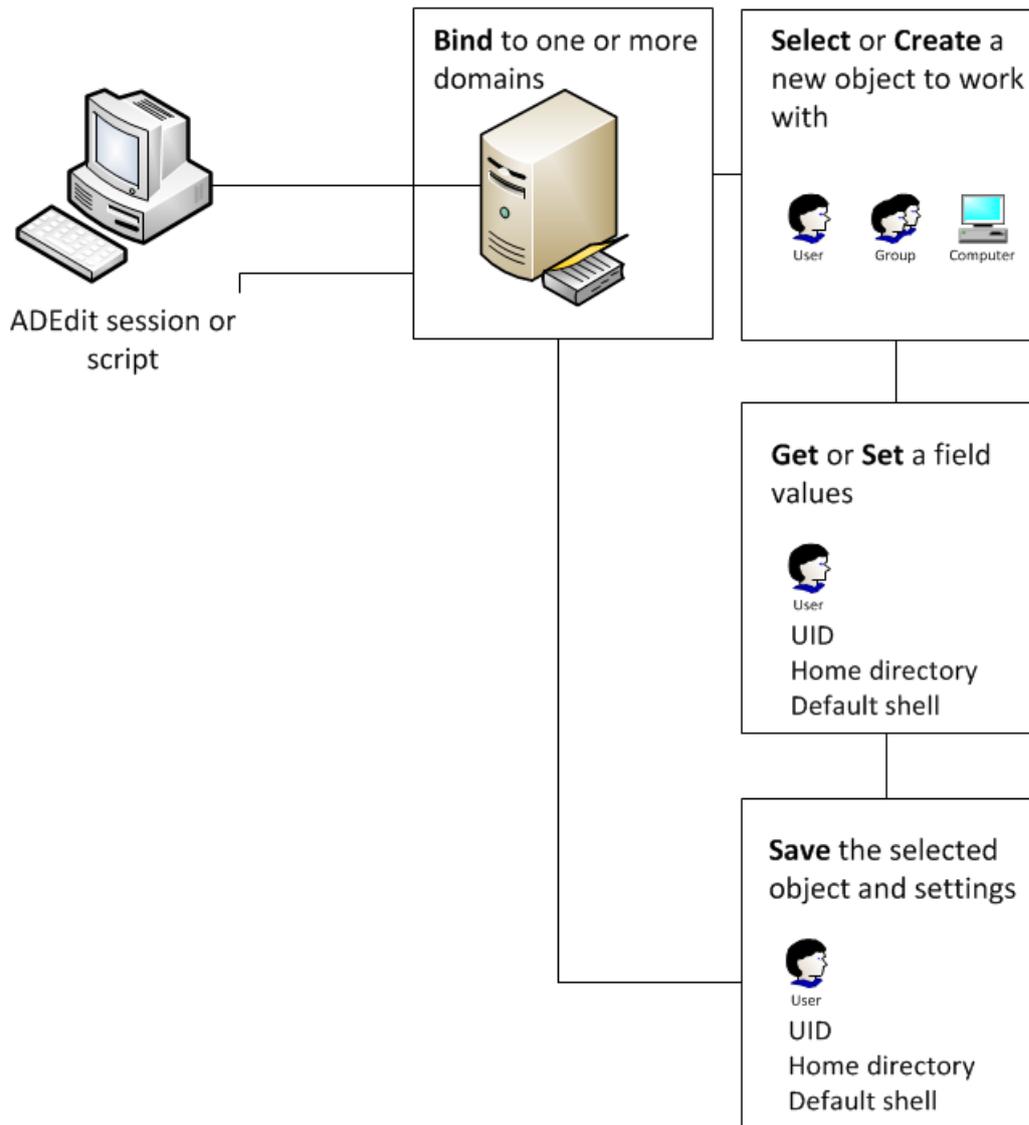
```
> help get*
```

## Learning to use ADEdit

You can use ADEdit interactively to run individual commands or to execute scripts directly. You can use ADEdit commands in scripts that you convert into executable files that can be execute outside of ADEdit sessions. Because scripts can automate and simplify many administrative tasks, it is important for you to know how to combine ADEdit commands in the proper sequence to get the results you are looking for.

Before you begin writing scripts that use ADEdit commands, you should be familiar with the most common logical flow for managing Centrify-specific and Active Directory objects.

The following illustration provides an overview of the logical process.



As illustrated, the typical logic flow in a ADEdit script follows these steps:

1. **Bind** ADEdit to one or more domains within a forest.  
The domains to which you bind will define the logical boundaries within which all subsequent commands work.
2. **Select** an existing Active Directory object or create a **new** object with which to work.

You can use `select` commands to retrieve existing object from Active Directory and store them in memory. You can use `new` commands to create new objects of a specified type and store them in the ADEdit context as the currently selected object.



There are also create commands that create a new objects in Active Directory without putting the object in the ADEdit context. You must explicitly select objects that are created with create commands.

3. **Get** or **set** values for a selected object.

After you select an object to work with and it is stored in memory—that is, the object is in the ADEdit context—you can read field values to see their current settings or write field values to change their current state.

4. **Save** the selected object and any settings you changed.

If you modify an object in memory or you have created a new object in memory, you must save it back to Active Directory for your changes to have any effect.

As these steps suggest, ADEdit is very context-oriented. The bindings you set and the objects you select determine the ADEdit current context. All commands work within that context. If you select a zone, for example, subsequent commands use the selected zone as the context in which to add new zone users, zone computers, and zone groups.

Outside of scripts that perform the most common administrative tasks, you might use ADEdit commands differently and without following these steps. For example, you might use ADEdit to convert data from one format to another, view help, or get information about the local computer without following the typical logic flow, but those tasks would be exceptions to the general rule.

## Binding to a domain and domain controller

ADEdit must bind to one or more domains before any ADEdit commands that affect Active Directory objects will work. When you execute the `bind` command, you specify the domain to which to bind. You can also specify a user name and password for the bind operation to provide authentication.

The domain can be any domain in the current forest. The ADEdit host computer does not have to be joined to a domain to bind to and work with a domain. A binding command can be as simple as:

```
>bind acme.com
```

If you specify a domain with no options, ADEdit automatically finds the domain's closest, fastest domain controller. Options can narrow down the

• • • • •

choice of domain controllers. The `-write` option, for example, specifies that you want ADEdit to choose a writable domain controller. The `-gc` option specifies that ADEdit use the global catalog (GC) domain controller. You can use both options to choose a writable GC domain controller, for example:

```
>bind -write -gc acme.com
```

Alternatively, you can name a specific domain controller as a part of the domain name:

```
>bind dcserv1@acme.com
```

**Note** Active Directory is a multi-master LDAP system. Changes made at any one domain controller eventually propagate to all other domain controllers in the domain (if they're universal changes). If any administration tools—such as Active Directory Users and Computers, Access Manager, or other instances of ADEdit—bind to the same domain controller, changes made by any one of the tools are immediately available to the other tools without waiting for propagation.

## Authentication

If no credentials are provided with a `bind` command, ADEdit gets its authentication data from the Kerberos credentials cache if one exists. Alternatively, you can provide a user name or both a user name and password. For example:

```
>bind acme.com administrator {e$t86&CG}
```

Notice that the password is enclosed in braces (`{}`) to ensure that Tcl handles it correctly. Without the braces, Tcl syntax will automatically substitute for some characters such as the `$` used in the password. For example, a dollar sign specifies the contents of a variable in Tcl. Enclosing a string in braces guarantees that Tcl will not try to substitute for any of the characters in the string. Tcl drops the braces when it passes the string on.

You can also use the credentials of the ADEdit's host computer by using the `-machine` option:

```
>bind -machine acme.com
```

**Note** Whatever credentials you use, they must be for an account on the Active Directory domain controller with enough authority to read from



**Note** and make changes to Active Directory objects in the domain. Without the proper authority, ADEdit commands that use Active Directory won't work.

## Binding scope and persistence

Binding to a single domain allows ADEdit commands to work on Active Directory in that domain. You can bind to multiple domains to allow ADEdit commands to work on more than one domain. To bind to multiple domains, you simply use multiple `bind` commands, one for each domain.

Once bound to a domain, ADEdit remains bound to that domain until another binding occurs to the same domain (possibly using a different authentication or specifying a different domain controller) or until the current interactive session or executing script ends. Binding might also end if the current context is popped and ADEdit reverts to an earlier context without the binding.

## Binding and join differences

The ADEdit `bind` operation is not the same as having the ADEdit host computer join an Active Directory domain. A join is the `adcli` connection to Active Directory for the host computer. A computer is only allowed to join one domain. A `bind` is an ADEdit connection to Active Directory, and it can be to more than one domain in the forest. The binding is completely independent of the host computer's joined domain.

**Note** A few ADEdit commands that start with `joined_*` use `adcli` to retrieve data from Active Directory. Those commands are affected by the host computers's joined domain because they require `adcli` to be connected to Active Directory and can only get data from the joined domain.

## Controlling binding operation

You can control the way ADEdit's binding to Active Directory operates. The `set_ldap_timeout` command sets a time interval for ADEdit's LDAP queries to execute by Active Directory. ADEdit considers a query that doesn't execute by the time-out interval as failed.



## Selecting an object

ADEdit manages Centrify information by working with the objects in Active Directory. The Centrify-specific object types are:

- Zones
- Zone users
- Zone computers
- Zone groups
- Roles
- Role assignments
- Privileged UNIX command rights
- PAM application rights
- NIS maps

However, you are not limited to using ADEdit only for managing Centrify-specific object types. You can also use ADEdit commands to work with generic Active Directory objects, including computers, users, groups, and other classes.

### Selection commands

The ADEdit object select commands have the form `select_xxx` where `xxx` is an object type. When you select an object (`select_zone`, for example), ADEdit looks for the object specified in Active Directory and retrieves it to store the object in the current context.

Each select command is tailored to the type of object it retrieves. As an example, after binding to `acme.com`, you can use a `get_zones` command to list the zones in the bound domain, then use a `select_zone` command to select the zone you want to work with:

```
>get_zones acme.com
{CN=default,CN=Zones,CN=Centrify,CN=Program
Data,DC=acme,DC=com}
{CN=cz1,CN=Zones,CN=Centrify,CN=Program
Data,DC=acme,DC=com}
{CN=cz2,CN=Zones,CN=Centrify,CN=Program
```

• • • • •

```
Data,DC=acme,DC=com}  
{CN=global,CN=Zones,CN=Centrify,CN=Program  
Data,DC=acme,DC=com}  
>select_zone {CN=global,CN=Zones,CN=Centrify,CN=Program  
Data,DC=acme,DC=com}
```

As this example illustrates, each zone is list by its distinguished name (DN) and you use the distinguished name to identify the zone you want to use.

## Selection as part of context

Once an object is selected, it resides in memory (context) with all attendant field values. Further ADEdit commands can examine and modify the object in context.

ADEdit keeps only one selected object of each type in context at a time. If you select or create another object of the same type, the new object replaces the old object in memory without saving the old object to Active Directory. ADEdit can and does keep multiple objects in context, but each object must be a different type.

**Note** A currently selected object often affects work on other objects types, especially the currently selected zone. For example, if you select a zone user, you must first select a zone so that ADEdit knows in which zone to look for the zone user. If you don't first select a zone, you can't select and work on various zone objects such as zone users, zone computers, and zone groups. Knowing your context as you work on objects is important.

## Persistence

A selected object stays selected until another object of the same type replaces it or until the current interactive session ends or executing script ends. When an ADEdit session ends, all selected objects are removed from ADEdit's memory. In most cases, you must explicitly save changes to objects in memory to ensure the changes are stored in Active Directory.



## Creating a new object

You can use ADEdit `new_xxx` commands, where `xxx` is the object type, to create new objects to work on instead of selecting existing objects. When you use `new_xxx` commands, ADEdit creates an object of the specified type and stores the object as the currently selected object of that type in ADEdit's current context.

In most cases, ADEdit does not provide default values for a new object's fields. If you create a new object, its fields are empty. You can use the ADEdit `set_xxx` commands to set values for the fields that are specific to each object type.

Here are some notes about creating objects in ADEdit:

- Creating a new zone works differently than all other object types: ADEdit does not create a new zone in memory. ADEdit creates new zones directly in Active Directory and fills in zone fields with default values. After you create a zone, you must then select it to examine and modify it.
- ADEdit cannot create AIX extended attributes in a Microsoft Services for UNIX (SFU) zone (Ref: CS-25392c).
- Some non-alphanumeric characters are valid for Windows user or group names and are converted to underscore ("\_") when changed to be UNIX names in the Access Manager, but cannot be used in `adedit`. (Ref: IN-90001) The following characters cannot be used in `adedit`: `\ ( ) + ; " , < > =`

## Examining objects and context

The ADEdit context is a combination of current bindings and currently selected objects. You can examine the properties of currently selected objects using ADEdit `get_xxx` or `list_xxx` commands, where `xxx` is an object type. For example, you can use the `get_roles` or `list_roles` command to see a list of roles in the current zone.

## Getting field values for objects

You can also use `get_xxx_field` commands to retrieve field values for different types of objects. For example:

```
>select_zone_user adam.avery@acme.com
```

.....

```
>get_zone_user_field uname  
adam
```

In this example, ADEdit retrieves the value of the field `uname`—in this case, the UNIX user name field—for the currently selected zone user `adam.avery@acme.com`.

## Getting current context information

You can examine ADEdit's current context at any time using two different commands: the `show` command and the `get_bind_info` command.

The `show` command returns all bindings and selected objects in the current context. For example:

```
>show  
Bindings:  
    acme.com: calla.acme.com  
Current zone:  
    CN=global,CN=Zones,CN=Centrify,CN=Program  
Data,DC=acme,DC=com  
Current nss user:  
    adam.avery@acme.com:adam:10001:10001:%  
{u:samaccountname}:%{home}/%{user}:%{shell}:
```

You can use optional arguments to limit the information the `show` command returns.

The `get_bind_info` command returns information about a bound domain. When you use this command, you specify the information you want to retrieve, such as the domain's forest, the name of the current domain controller, the domain's security identifier (SID), the functional level of the domain, or the functional level of the domain's forest. For example:

```
>get_bind_info acme.com server  
adserve02.acme.com
```

In this case, ADEdit returns the name of the bound server for the domain `acme.com`.

.....

## Modifying or deleting selected objects

Once an object is selected and residing in the ADEdit context, you can modify its fields using the ADEdit `set_xxx_field` commands, where `xxx` is the object type. These commands allow you to specify a field name and a field value. For example:

```
>select_zone_user adam.avery@acme.com  
>set_zone_user_field uname buzz
```

This example selects the zone user `adam.avery@acme.com` and sets the `uname` field for the zone user—the UNIX user name—to `buzz`. The field is set to the new value only in memory, however. You must save the object before the new field value is stored in Active Directory and takes effect within the object's domain. For example:

```
>save_zone_user
```

## Deleting an object

You can delete a currently selected object using the ADEdit `delete_xxx` commands, where `xxx` is the object type. When you delete an object, it is deleted from both memory *and* Active Directory. For example:

```
>select_zone_user adam.avery@acme.com  
>delete_zone_user
```

This example deletes the currently selected zone user, `adam.avery@acme.com`, from the ADEdit context so there's no longer a selected zone user. The command also deletes the zone user object associated with the user `adam.avery@acme.com` so there's no longer a zone user by that name in Active Directory.

**Note** There is no undo for a delete command. Once the object is deleted from Active Directory, you must recreate it if you want it back. Be especially careful if you set up an ADEdit script to delete multiple objects.



## Saving selected objects

Any new or modified object in ADEdit's context has no effect until you save the object back to Active Directory. You do so using a `save_xxx` command where `xxx` is the object type. For example:

```
>save_zone
```

This example saves the currently selected zone object back to Active Directory along with any field values that have been modified since the zone was selected.

Saving an object does not deselect the object. It remains the selected object in memory so that you can further read and modify it.

## Pushing and popping context

There are times when you may want to save ADEdit's current context, change it to a new context to work on different objects in different domains, and then revert back to the original context. This is particularly true when writing Tcl scripts with subroutines, where you may want to feel free to complete a completely new context without altering the context of the calling code.

ADEdit offers a `push` and a `pop` command to save and retrieve contexts to a stack maintained in memory. `push` saves the complete current context—all of its bindings and selected objects—to the stack. Subsequent `push` commands save more contexts to the top of the stack, pushing the older contexts further down the stack, allowing for nested subroutines.

`pop` reads the context from the top of the stack and restores it to memory as the current context. `pop` also removes the restored context from the stack. Subsequent `pop` commands pop more contexts off the stack until the stack is empty, at which point `pop` returns an error.

## Creating ADEdit scripts

You can combine ADEdit commands into scripts that perform many common administrative tasks, such as creating new zones, adding users to zones, or pre-creating computer accounts. After you create a script, you can execute it



from a shell that calls `adedit` or convert it to an executable file that can run directly from the command line.

## Starting with a simple script

If you are new to scripting, Tcl, or both, you might want to experiment first with a few simple commands before trying to develop scripts that perform administrative tasks. The steps in this section are intended to help you get started.

If you are already familiar with scripting languages or with using Tcl, you might want to skip ahead to the discussion of the sample scripts or directly to the command reference.

### To write a simple ADEdit script:

1. Open a new file—for example, `my_adedit_script`—in a text editor.
2. Type the following line to set up the `adedit` environment and include the ADEdit Tcl library:

```
#!/bin/env adedit
package require ade_lib
```

If your version of Linux or UNIX has the `env` command in a location other than the `/bin` directory, modify the first line to specify that directory. For example, another common location for the `env` command is `/usr/bin`. In this case, you would type:

```
#!/usr/bin/env adedit
```

3. Type an appropriate `bind` command to identify the Active Directory domain or domains to use.

```
bind pistols.org maya.garcia {$m113s88}
```

Depending on whether you are going to run this script interactively or as an executable file, you might include or exclude authentication information.

4. Type the appropriate commands to create and select a new zone.

```
create_zone tree
"cn=sample,cn=zones,ou=centrify,dc=acme,dc=com" std
```

.....

```
select_zone  
"cn=sample,cn=zones,ou=centrify,dc=acme,dc=com"
```

5. Type the command to list the current zones to stdout to verify the new zone.

```
list_zones pistols.org
```

6. Type the command to save the zone and quit.

```
save_zone  
quit
```

7. Save the text file and execute it using ADEdit or as an executable file.

After you have tested the basic script, you edit it to create new zones, make a zone a child zone, add new zone computers, groups, or users. for example, you might add lines similar to these:

```
new_zone_user AD_user_UPN  
set_zone_user_field field value  
save_zone_user  
list_zone_users
```

If your sample script creates and selects a zone successfully, you should delete or rename the zone each time you iterate through the execution.

The following is a sample of what the simple script might look like:

```
#!/bin/env adeditpackage require ade_lib  
bind pistols.org maya.garcia {$m113s88}  
create_zone tree  
"cn=test6,cn=zones,ou=centrify,dc=pistolas,dc=org" std  
select_zone  
"cn=test6,cn=zones,ou=centrify,dc=pistolas,dc=org"  
set_zone_field parent "cn=US-  
HQ,cn=zones,ou=centrify,dc=pistolas,dc=org"  
list_zones pistols.org  
save_zone  
new_zone_user tim@pistolas.org  
set_zone_user_field uname tim  
set_zone_user_field uid 81000  
set_zone_user_field gid 81000  
set_zone_user_field gecos "Tim Jackson, Accounting"  
save_zone_user  
list_zone_users  
quit
```



## Executing an ADEdit script using ADEdit

You can execute ADEdit script by invoking ADEdit on the command line or by making the script an executable file and invoking the script itself directly from the command line.

### To execute an ADEdit script by invoking ADEdit on the command line:

1. Open a shell.
2. Type `adedit` followed by the name of the script  
For example, if the name of the script is `my_adedit_script` and it is the current working directory, type:

```
adedit my_adedit_script
```

If the script isn't in the current working directory, specify the path to the script and any arguments if the script requires any.

## Running an ADEdit script as an executable from the command line

You can run an ADEdit script without invoking ADEdit first by making the script an executable file.

### To run an ADEdit script as a UNIX-executable file:

1. Verify the script begins with the following lines:

```
#!/bin/env adedit  
package require ade_lib
```

The script reads it as a comment, however UNIX or Linux will use it to find and execute ADEdit and then execute the rest of the script.

2. Use `chmod` to make the file executable.

For example, if the name of the script is `my_adedit_script` and it is the current working directory, type:

```
chmod +x my_adedit_script
```

• • • • •

3. Make sure the file's directory is listed in your PATH environment variable if you want to be able to execute the file from any directory.

Alternatively, modify the script to include the full path to adedit. For example:

```
#!/bin/env /usr/bin/adedit
```

Once set up this way, you can simply enter the script's file name in a shell and have the script execute as a command.

```
/my_adedit_script
```

### Running an ADEdit script as a shell script

You can also run the script as a shell script. In this case, the script file would have the .sh suffix and would contain the following lines at the beginning of the file:

```
#!/bin/sh
# \
exec adedit "$0" ${1+"$@"}
package require ade_lib
```

# ADEdit commands organized by type

As discussed in [Logical organization for ADEdit commands](#), there are different types of ADEdit commands that can be organized into logical categories. This chapter provides a brief introduction to the ADEdit commands in each of those logical categories. For detailed information about individual commands, see [ADEdit command reference](#).

## General-purpose commands

You can use the following general-purpose commands to control overall ADEdit operation or return general information about ADEdit or its host computer.

Command	Description
<code>help</code>	Returns information about a specified ADEdit command or all ADEdit commands.
<code>get_adinfo</code>	Returns information about the joined domain, the joined zone, or the name the local computer is joined under.
<code>quit</code>	Quits ADEdit.
<code>set_ldap_timeout</code>	Sets the time-out value used by ADEdit's LDAP commands that perform read and write operations on Active Directory through a binding.

## Context commands

You can use the following context commands set the current domain bindings, report on the current bindings and selected object, and save and



retrieve the ADEdit context (which includes both bindings and currently selected objects).

Command	Description
<code>bind</code>	Binds to one or more Active Directory domains to define the ADEdit context for subsequent commands.
<code>get_bind_info</code>	Returns information about the domains to which ADEdit is bound.
<code>pop</code>	Restores the context from the top of the ADEdit context stack.
<code>push</code>	Saves the current context to the ADEdit context stack.
<code>show</code>	Displays the current context of ADEdit, including its bound domains and currently selected objects.
<code>validate_license</code>	Determines whether there is a valid license and stores an indicator in the ADEdit context.

## Object-management commands

You can use object-management command to retrieve, modify, create, and delete Active Directory objects of any kind, including Centrify-specific objects such as zones, rights, and roles. The command set for each object type is similar to the command sets for the other object types.

### Zone object management commands

You can use the following zone object management commands to create, select, save, and delete zones and manage zone properties.

Command	Description
<code>create_zone</code>	Creates a new zone in Active Directory.
<code>delegate_zone_right</code>	Delegates a zone administrative task to a specified user or group.
<code>delete_zone</code>	Deletes the selected zone from Active Directory and memory.
<code>get_child_zones</code>	Returns a Tcl list of child zones, computer roles, or computer-specific zones associated with the current zone.
<code>get_zone_field</code>	Returns the value for a specified field from the currently selected zone.
<code>get_zone_nss_vars</code>	Returns the NSS substitution variable for the selected zone.



Command	Description
<code>get_zones</code>	Returns a Tcl list of all zones within a specified domain.
<code>save_zone</code>	Saves the selected zone with its current settings to Active Directory.
<code>select_zone</code>	Retrieves a zone from Active Directory and stores it in memory as the currently selected zone.
<code>set_zone_field</code>	Sets the value for a specified field in the currently selected zone.

## Zone user object management commands

You can use the following zone user commands to create, select, save, and delete zone user objects and manage user properties in the currently selected zone.

Command	Description
<code>delete_local_user_profile</code>	Deletes a local user (that is not an Active Directory user) that has a profile defined in the current zone.
<code>delete_zone_user</code>	Deletes the zone user from Active Directory and from memory.
<code>get_local_user_profile_field</code>	Returns the value of a profile field for the currently selected local user (that is not an Active Directory user) that has a profile defined in the current zone.
<code>get_local_users_profile</code>	Returns a Tcl list of profiles for local users (that are not Active Directory users) that are defined in the currently selected zone.
<code>get_zone_user_field</code>	Returns the value for a specified field from the currently selected zone user.
<code>get_zone_users</code>	Returns a Tcl list of the Active Directory names of zone users in the current zone.
<code>list_local_users_profile</code>	Returns a list of local users (that are not Active Directory users) that have a profile defined in the current zone.
<code>list_zone_users</code>	Lists all zone users with NSS data for each user in <code>stdout</code> .
<code>new_local_user_profile</code>	Creates an object for a local user (that is not an Active Directory user) in the currently selected zone.
<code>new_zone_user</code>	Creates a new zone user and stores it in memory as the currently selected zone user.



Command	Description
<code>save_local_user_profile</code>	Saves the object for the currently selected local user (that is not an Active Directory user) after you create the local user object or edit profile field values for the local user object.
<code>save_zone_user</code>	Saves the selected zone user with its current settings to Active Directory.
<code>select_local_user_profile</code>	Selects a local user (that is not an Active Directory user) object for viewing or editing.
<code>select_zone_user</code>	Retrieves a zone user from Active Directory and stores it in memory as the selected zone user.
<code>set_local_user_profile_field</code>	Sets the value of a field for the currently selected local user (that is not an Active Directory user) that has a profile defined in the current zone.
<code>set_zone_user_field</code>	Sets the value for a specified field in the currently selected zone user.

## Zone group object management commands

You can use the following zone group commands to create, select, save, and delete zone group objects and manage group properties in the currently selected zone.

Command	Description
<code>delete_local_group_profile</code>	Deletes a local group (that is not an Active Directory group) that has a profile defined in the current zone.
<code>delete_zone_group</code>	Deletes the zone group from Active Directory and from memory.
<code>get_local_group_profile_field</code>	Returns the value of a profile field for the currently selected local group (that is not an Active Directory group) that has a profile defined in the current zone.
<code>get_local_groups_profile</code>	Returns a Tcl list of profiles for local groups (that are not Active Directory groups) that are defined in the currently selected zone.
<code>get_zone_group_field</code>	Returns the value for a specified field from the currently selected zone group.
<code>get_zone_group</code>	Return a Tcl list of Active Directory names of all zone groups in the current zone.



Command	Description
<code>groups</code>	
<code>list_local_groups_profile</code>	Returns a list of local groups (that are not Active Directory groups) that have a profile defined in the current zone.
<code>list_zone_groups</code>	Lists all zone groups with object data for each group in <code>stdout</code> .
<code>new_local_group_profile</code>	Creates an object for a local group (that is not an Active Directory group) in the currently selected zone.
<code>new_zone_group</code>	Creates a new zone group and stores it in memory as the currently selected zone group.
<code>save_local_group_profile</code>	Saves the object for the currently selected local group (that is not an Active Directory group) after you create the local group object or edit profile field values for the local group object.
<code>save_zone_group</code>	Saves the selected zone group with its current settings to Active Directory.
<code>select_local_group_profile</code>	Selects a local group (that is not an Active Directory group) object for viewing or editing.
<code>select_zone_group</code>	Retrieves a zone group from Active Directory and stores it in memory as the selected zone group.
<code>set_local_group_profile_field</code>	Sets the value of a field for the currently selected local group (that is not an Active Directory group) that has a profile defined in the current zone.
<code>set_zone_group_field</code>	Sets the value for a specified field in the currently selected zone group.

## Zone computer object management commands

You can use the following zone computer commands to create, select, save, and delete zone group objects and manage computer properties in the currently selected zone.

Command	Description
<code>delete_zone_computer</code>	Deletes the zone computer from Active Directory and from memory.
<code>get_zone_</code>	Returns the value for a specified field from the currently selected zone



Command	Description
<code>computer_field</code>	computer.
<code>get_zone_computers</code>	Returns a Tcl list of Active Directory names of all zone computers in the current zone.
<code>list_zone_computers</code>	Lists all zone computers along with object data for each computer in <b>stdout</b> .
<code>new_zone_computer</code>	Creates a new zone computer and stores it in memory as the currently selected zone computer.
<code>save_zone_computer</code>	Saves the selected zone computer with its current settings to Active Directory.
<code>select_zone_computer</code>	Retrieves a zone computer from Active Directory and stores it in memory as the selected zone computer.
<code>set_zone_computer_field</code>	Sets the value for a specified field in the currently selected zone computer.

## Computer role object management commands

You can use the following computer role commands to create, select, save, and delete computer role objects and manage computer role properties in the currently selected zone.

Command	Description
<code>create_computer_role</code>	Creates a new computer role in Active Directory.
<code>delete_zone</code>	Deletes the selected computer role from Active Directory and memory.
<code>get_role_assignments</code>	Returns a Tcl list of user role assignments associated with the selected computer role.
<code>get_zone_field</code>	Retrieves the computer group associated with the computer role.
<code>list_role_assignments</code>	Lists user role assignments associated with the selected computer role.
<code>new_role_assignment</code>	Creates a new role assignment and associates it with the selected computer role.
<code>save_zone</code>	Saves the selected computer role with its current settings to Active Directory.
<code>select_zone</code>	Retrieves a computer role from Active Directory and stores it in memory as the selected zone for subsequent commands.
<code>set_zone_field</code>	Sets the computer group which is associated with the computer role.



## Role object management commands

You can use the following role object commands to create, select, save, and delete role objects and manage role properties in the currently selected zone.

Command	Description
<code>add_command_to_role</code>	Adds a privileged command to the currently selected role.
<code>add_pamapp_to_role</code>	Adds a PAM application right to the currently selected role.
<code>delete_role</code>	Deletes the selected role from Active Directory and from memory.
<code>get_role_apps</code>	Returns a Tcl list of the PAM applications associated with the currently selected role.
<code>get_role_commands</code>	Returns a Tcl list of the privileged commands associated with the currently selected role.
<code>get_role_field</code>	Returns the value for a specified field from the currently selected role.
<code>get_roles</code>	Returns a Tcl list of roles in the current zone.
<code>list_role_rights</code>	List all privileged commands and PAM applications associated with the currently selected role in <code>stdout</code> .
<code>list_roles</code>	Lists all roles in the currently selected zone along with object data for each role in <code>stdout</code> .
<code>new_role</code>	Creates a new role and stores it in memory as the currently selected role.
<code>remove_command_from_role</code>	Removes a privileged command from the currently selected role.
<code>remove_pamapp_from_role</code>	Removes a PAM application from the currently selected role.
<code>save_role</code>	Saves the selected role with its current settings to Active Directory.
<code>select_role</code>	Retrieves a role from Active Directory and stores it in memory as the selected role.
<code>set_role_field</code>	Sets the value for a specified field in the currently selected role.

## Role assignment object management commands

You can use the following role assignment object commands to create, select, save, and delete role assignment objects and manage role assignment properties in the currently selected zone.



Command	Description
<code>delete_role_assignment</code>	Deletes the selected role assignment from Active Directory and from memory.
<code>get_role_assignment_field</code>	Returns the value for a specified field from the currently selected role assignment.
<code>get_role_assignments</code>	Returns a Tcl list of role assignments in the current zone.
<code>list_role_assignments</code>	Lists all role assignments along with object data for each role assignment in <b>stdout</b> .
<code>new_role_assignment</code>	Creates a new role assignment and stores it in memory as the currently selected role assignment.
<code>save_role_assignment</code>	Saves the selected role assignment with its current settings to Active Directory.
<code>select_role_assignment</code>	Retrieves a role assignment from Active Directory and stores it in memory as the selected role assignment.
<code>set_role_assignment_field</code>	Sets the value for a specified field in the currently selected role assignment.

## PAM application object management commands

You can use the following PAM application commands to create, select, save, and delete PAM application objects and manage PAM application properties in the currently selected zone.

Command	Description
<code>delete_pam_app</code>	Deletes the selected PAM application from Active Directory and from memory.
<code>get_pam_apps</code>	Returns a Tcl list of PAM applications in the current zone.
<code>get_pam_field</code>	Returns the value for a specified field from the currently selected PAM application.
<code>list_pam_apps</code>	List all PAM applications along with object data for each PAM application in <b>stdout</b> .
<code>new_pam_app</code>	Creates a new PAM application and stores it in memory as the currently selected PAM application.
<code>save_pam_app</code>	Saves the selected PAM application with its current settings to Active Directory.
<code>select_pam_app</code>	Retrieves a PAM application from Active Directory and stores it in memory as the



Command	Description
<code>app</code>	selected PAM application.
<code>set_pam_field</code>	Sets the value for a specified field in the currently selected PAM application.

## Command (dz) object management commands

You can use the following privileged authorization commands to create, select, save, and delete privileged UNIX command and manage command properties in the currently selected zone.

Command	Description
<code>delete_dz_command</code>	Deletes the selected command from Active Directory and from memory.
<code>get_dz_commands</code>	Return a Tcl list of commands in the current zone.
<code>get_dzc_field</code>	Returns the value for a specified field from the currently selected command.
<code>list_dz_commands</code>	List all privileged commands along with object data for each command in <code>stdout</code> .
<code>new_dz_command</code>	Creates a new command and stores it in memory as the currently selected command.
<code>save_dz_command</code>	Saves the selected command with its current settings to Active Directory.
<code>select_dz_command</code>	Retrieve a privileged command from Active Directory and stores it in memory as the selected command.
<code>set_dzc_field</code>	Sets the value for a specified field in the currently selected command.

## NIS map object management commands

You can use the following NIS map commands to create, select, save, and delete NIS maps and manage NIS map entries and properties in the currently selected zone.

Command	Description
<code>add_map_entry</code>	Adds an entry to the currently selected NIS map.
<code>add_map_entry_with_comment</code>	Adds an entry with comments to the currently selected NIS map.



Command	Description
<code>delete_map_entry</code>	Removes an entry from the currently selected NIS map.
<code>delete_nis_map</code>	Deletes the selected NIS map from Active Directory and from memory.
<code>get_nis_map</code>	Returns a Tcl list of the entries in the currently selected NIS map.
<code>get_nis_map_field</code>	Returns the value for a specified field from the currently selected NIS map.
<code>get_nis_map_with_comment</code>	Returns a Tcl list of the entries with their comments in the currently selected NIS map.
<code>get_nis_maps</code>	Returns a Tcl list of NIS maps in the current zone.
<code>list_nis_map</code>	Lists the NIS map entries from the currently selected NIS map in <b>stdout</b> .
<code>list_nis_map_with_comment</code>	Lists the NIS map entries and comments from the currently selected NIS map in <b>stdout</b> .
<code>list_nis_maps</code>	List all NIS maps in the currently selected zone in <b>stdout</b> .
<code>new_nis_map</code>	Creates a new NIS map and stores it in memory as the currently selected NIS map.
<code>save_nis_map</code>	Saves the selected NIS map with its current entries to Active Directory.
<code>select_nis_map</code>	Retrieves a NIS map from Active Directory and stores it in memory as the selected NIS map.

## Active Directory object management commands

You can use the following Active Directory commands to create, select, save, and delete NIS maps and manage NIS map entries and properties in the currently selected zone.

Command	Description
<code>add_object_value</code>	Adds a value to a multi-valued field attribute of the currently selected Active Directory object.
<code>delete_object</code>	Deletes the selected Active Directory object from Active Directory and from memory.
<code>delete_sub_tree</code>	Deletes an Active Directory object and all of its children.
<code>get_object_field</code>	Returns the value for a specified field from the currently selected Active Directory object.
<code>get_object_field_names</code>	Returns a Tcl list of the field names for each of the fields attributes associated the currently selected Active Directory object.



Command	Description
<code>get_objects</code>	Performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of matching objects.
<code>new_object</code>	Creates a new Active Directory object and stores it in memory as the currently selected Active Directory object.
<code>remove_object_value</code>	Removes a value from a multi-valued field attribute of the currently selected Active Directory object.
<code>save_object</code>	Saves the selected Active Directory object with its current settings to Active Directory.
<code>select_object</code>	Retrieves an object with its attributes from Active Directory and stores it in memory as the selected Active Directory object.
<code>set_object_field</code>	Sets the value for a specified field in the currently selected Active Directory object.

## Utility commands

You can use the following utility commands retrieve and convert data from format to format, manipulate distinguished names, and manage group membership and user passwords.

Command	Description
<code>dn_from_domain</code>	Converts a domain's dotted name to a distinguished name (DN) format.
<code>dn_to_principal</code>	Searches Active Directory for a DN and, if found, returns the corresponding UPN.
<code>domain_from_dn</code>	Converts a domain's distinguished name (DN) to a dotted name format.
<code>get_group_members</code>	Returns a Tcl list of members in a group.
<code>get_parent_dn</code>	Returns the parent of an LDAP path (a distinguished name): it removes the first element of the DN and returns the rest.
<code>get_pwnam</code>	Searches the <code>etc/passwd</code> file for a UNIX user name and, if found, returns a Tcl list of the <code>passwd</code> profile values associated with the user.
<code>get_rdn</code>	Returns the relative DN of an LDAP path: it returns only the first element of the supplied DN.
<code>get_schema_guid</code>	finds a class or attribute in Active Directory and returns its globally unique identifier (GUID)
<code>getent_passwd</code>	Returns a Tcl list of all entries in the local <code>/etc/passwd</code> file.



Command	Description
<code>joined_get_user_membership</code>	Uses addclient to query Active Directory and returns a Tcl list of groups that a user belongs to.
<code>joined_name_to_principal</code>	Uses addclient to search for a UNIX name and return the security principal associated with that UNIX name.
<code>joined_user_in_group</code>	Uses addclient to check Active Directory to see if a user is in a group.
<code>move_object</code>	Moves the selected object to the specified location.
<code>principal_from_sid</code>	Searches Active Directory for an SID and returns the security principal associated with the SID.
<code>principal_to_dn</code>	Searches Active Directory for a user principal name (UPN) and, if found, returns the corresponding DN.
<code>rename_object</code>	Renames the selected object.
<code>set_user_password</code>	Sets an Active Directory user's password.
<code>sid_to_escaped_string</code>	Converts an Active Directory security identifier (SID) to an escaped string.
<code>sid_to_uid</code>	Converts an Active Directory SID to a user ID (UID).

## Security descriptor commands

You can use the following security descriptor commands modify SDs and make them readable by humans.

Command	Description
<code>add_sd_ace</code>	Adds an access control entry to a security descriptor.
<code>explain_sd</code>	Converts a security description in SDDL format to a human-readable form.
<code>remove_sd_ace</code>	Removes an access control entry (ACE) from a security descriptor.
<code>set_sd_owner</code>	Sets the owner of a security descriptor.

# Using the demonstration scripts

This chapter describes the ADEdit sample scripts provided in the package. The scripts are listed in the following table. The corresponding source files are in the `/usr/share/centrifdc/samples/adedit` directory. The source file name is shown in the table and each script header.

You have a couple of different ways to invoke scripts from the command line (see [Creating ADEdit scripts](#)). The sample scripts demonstrate two of them.

Section heading	Purpose	Source file name
Reading command line input	These scripts illustrate two different methods for using the Tcl <code>argv</code> , <code>argc</code> , and <code>argv0</code> variables.	<code>MktDept.sh</code> <code>getopt-example</code>
Create a parent zone	This script illustrates how to create a Centrifly parent zone.	<code>CreateParentZone</code>
Create child zones	This script illustrates how to create two child zones in a parent zone.	<code>CreateChildZones</code>
Create privileged commands and roles	These scripts illustrate how to create new privileged commands and new roles that include those commands.	<code>MakeRole</code> <code>ApacheAdminRole</code>
Add and provision UNIX users	This script and input file illustrate how to add users to Active Directory and copy them to the Active Directory UNIX <code>Users</code> group.  If you have the Zone Provision Agent configured and running, you can use this script or one similar to it to automatically provision user profiles when users are added to Active Directory.	<code>AddUnixUsers</code> <code>users.txt</code>
Simple tools	These scripts demonstrate how you can list the	<code>computers-report</code>



Section heading	Purpose	Source file name
	computers in a zone, extract field attributes from user objects, and list the users in a zone.	useracc-report user-report GetComputers
Run a script from a script	These scripts illustrate how you can call a script ( <code>setenv</code> ) from within another script to perform different queries based on the values entered.	setenv GetChildZones GetGroups GetUsers GetZones

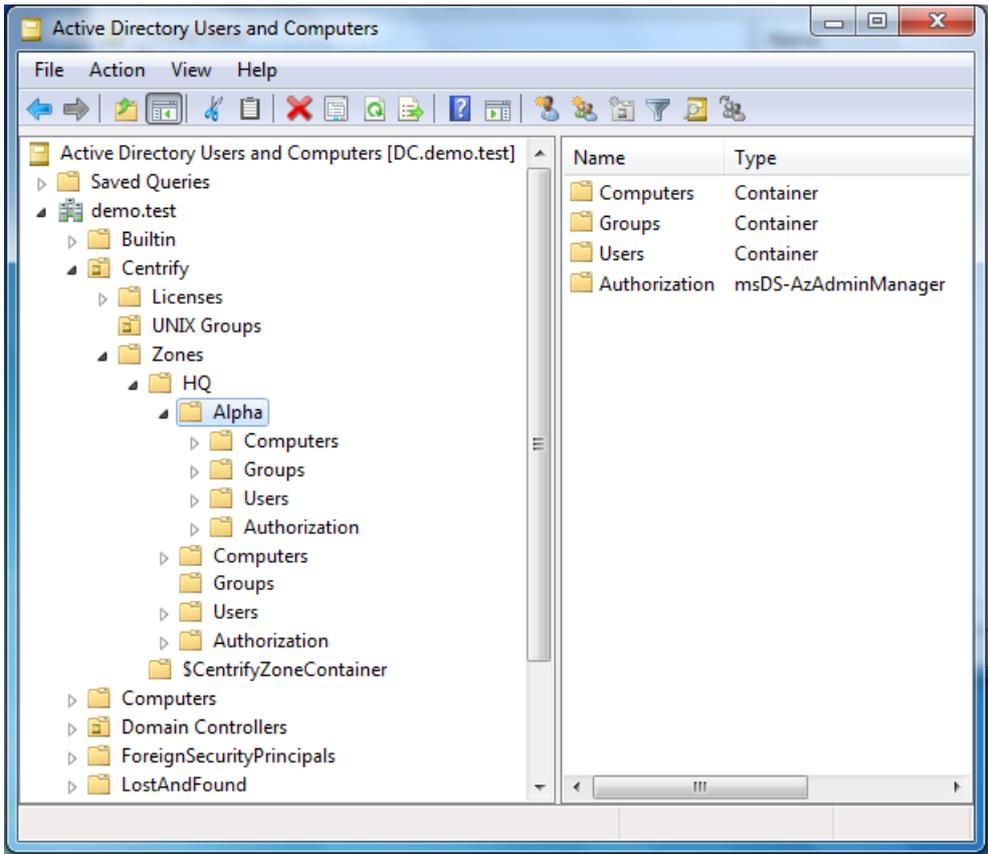
## Zone containers and nodes

Many ADEdit commands require you to specify the zone container. This container is the root container used by Centrify to store the zone information for the users, groups, computers and child zones. This container can have any name and can be anywhere in Active Directory. This container can also be an organizational unit.

Before you proceed, you need to know the location of the zone containers in Active Directory and the distinguished names you use to specify the zone container and its objects.

This section illustrates some sample cases with different locations for the zone container and the distinguished name for commonly used variables in the scripts.

In this example, the installer defined a base organizational unit called `centrify`. This architecture is often used because it puts all the UNIX-related information in a single branch. The container with the zone information is called `Zones`.



In addition to the zones container location, the installation script requires the installer to specify a location for a container to store the Centrifly software licenses. In this figure, the node—Licenses—is in the base organizational unit. However, it does not need to be there.

In this figure, the installer also created another organizational unit called UNIX Groups for the Active Directory groups used for the UNIX users. Keeping all of the groups recreated for the UNIX users in a single node simplifies managing them and the privileges assigned to each user. (With few exceptions, the UNIX users get their rights from the role assigned to the group in which they are a member.) Often, more organizational units are created for managing different classes of UNIX user and UNIX services.

There are two zones in this figure: the parent zone HQ and a child zone named Alpha. Each zone contains nodes labeled Computers, Groups, users, and Authorization. When you specify a zone, computer, user, or group in an ADEdit command you must use the distinguished name. The following table illustrates the distinguished names.



Object type	Example	Example distinguished name
Domain	demo.test	dc=demo,dc=test
Base organizational unit	Centrifly	ou=Centrifly,dc=demo,dc=test
Zone container	Zones	cn=Zones,ou=Centrifly,dc=demo,dc=test
Parent zone	HQ	cn=HQ,cn=Zones,ou=Centrifly,dc=demo,dc=test
Child zone	Alpha	cn=Alpha,cn=HQ,cn=Zones,ou=Centrifly,dc=demo,dc=test
Organizational unit	UNIX Groups	“ou=UNIX Groups,ou=Centrifly,dc=demo,dc=test”
UNIX group	ApacheAdmins	“cn=ApacheAdmin,ou=UNIX Groups,ou=Centrifly,dc=demo,dc=test”
Computer in Alpha zone	RHEL	cn=RHEL,cn=Computers,cn=Alpha,cn=HQ,cn=Zones,ou=Centrifly,dc=demo,dc=test

You should note that distinguished names can contain space, as illustrated by the UNIX Groups organizational unit. To prevent Tcl from interpreting a space as new element in a list, you can enclose the distinguished name with double quotes (“ ”) or using braces ({}). When specifying distinguished names, you should also be sure to use ou and cn correctly. Commands will fail if you refer to an organizational unit using cn.

## Create Tcl procedures

The following example demonstrates how to create procedures using the Tcl proc command. These two procedures create a new Active Directory user and Active Directory group, respectively, but first check to see if that object already exists in Active Directory.

This example uses the Tcl catch and if commands to determine if the account already exists. catch takes a Tcl script (in this case, the select\_ object command) and returns a 1 if an error (in this case, the account does NOT exist) occurs. Inside the if command, a non-zero result of the expression causes the body commands (puts and create\_aduser or create\_adgroup) to

.....

be executed. Otherwise, if `select_object` is successful (the account exists) it does not create the new account.

**Note** See the `Addunixusers` script for a similar example that uses the `catch` and `if` commands to determine if a user exists.

## Create Active Directory group procedure

```
# The following procedure creates an Active Directory group
if a
# group with the same distinguished name does not already
exist.
proc my_create_adgroup {dn sam gtype} {
    if { [catch {select_object $dn}] } {
        # If we fail to select the object, the group
        # does not exist. So we create it here.
        puts "Creating $dn"
        create_adgroup $dn $sam $gtype
    } else {
        puts "$dn exists. Do not create."
    }
}
```

## Create Active Directory user procedure

```
# The following procedure creates an Active Directory user
if an
# account with the same distinguished name does not already
exist.
proc my_create_aduser {dn upn sam pw} {
    if { [catch {select_object $dn}] } {
        # If we fail to select the object, the account
        # does not exist. So we create it here.
        puts "Creating $dn"
        create_aduser $dn $upn $sam $pwd
    } else {
        puts "$dn exists. Do not create."
    }
}
```

.....

## Reading command line input

In general, Tcl reads the arguments following the script name as a list and creates the following three variables:

- `argv`: A Tcl list containing all of the arguments in the command line
- `argc`: A count of the number of arguments in the lists
- `argv0`: The script name.

For example, the following script uses all three variables. This is a simple command in the form

```
>/bin/sh MktDept.sh name name name
```

where *name* is a person's name, such as Mary or Joe. If you want to use first and last name, surround the name with quotes, for example "Jane Smith".

**Note** This code sample demonstrates starting ADEdit from a shell script. The subsequent examples use the executable file model.

### MktDept.sh

```
#!/bin/sh
# This script takes a list of names and displays it
#
# \
exec adedit "$0" ${1+"$@"}
package require ade_lib
if { $argc == 0 } {
    puts "Command format: $argv0 name name ..."
    exit 1
}
set total $argc
puts "
The following people are in the marketing department"
while { $total > 0 } {
    incr total -1
    puts "[lindex $argv $total]"
}
```

The first `if` statement uses the count, `argc`, to determine if any arguments have been entered. If the `argc` value is equal to zero, the user did not enter any names and the script displays the command format message. The `argc`

.....

counter is used again to set the total count of names entered for the while loop. Inside the loop, the names are drawn from the argv list.

Another useful command for parsing command line options is `getopt`. This command derives from, but is different than, the Tcl `getopt` command. The ADEdit `getopt` command has the following syntax:

```
getopt _argv name ?-var?
```

where:

- `_argv` is the Tcl list that contains the command line arguments.
- `name` is a label for the associated data.
- `?_var?` is the variable name for the data.

For example, the following script illustrates the use of `getopt` to define the user and group variables that will be used later in the script.

This script also demonstrates how to use a procedure, `usage`, that prompts the user when she doesn't enter all of the arguments. `usage` first displays the full command syntax and then the missing argument.

**Note** The user and password arguments are optional. If the user enters a user name without the password, the `bind` program automatically prompts for the password. You do not need to include that prompt in the script.

## getopt-example

```
#!/bin/env adedit
# This script takes a domain name and optionally user name
and password
# and binds the user to the specified domain.
# If the user does not specify a user name or password, she
is prompted.
#
package require ade_lib
proc usage {msg} {
    puts {usage: -d <domain> [-u <user>] [-p
<password>]}
    puts $msg
    exit 1
}
```

.....

```
if {[getopt argv -d domain] == 0} {
    usage "Missing Domain, ex. centrifify.demo"
}
if {[getopt argv -u user] != 0} {
    if {[getopt argv -p password]} {
        bind $domain $user $password
    } else {
        bind $domain $user}
    } else {
    puts "Enter administrator name:"
    gets stdin user
    bind $domain $user
}
puts "
Binding complete to $domain."
```

## Create a parent zone

This sample script illustrates how you can create a parent zone. This script uses the `puts` command to display information and to prompt the user to specify variables that will be used to create the parent zone object. The command line syntax is as follows:

```
>./CreateParentZone - z parentZone -u adminName [-p
password]
```

where:

- *parentZone* is the name of the parent zone you want to create.
- *adminName* is the name of an Active Directory user with administrator privileges on the domain controller.
- *password* is the administrator's password. If you do not enter the password in the command line, you are prompted to enter it.

Note that this sample script assumes you are using the default deployment structure with the top-level organizational unit. If you are not using the default deployment structure, you should modify the sample script to reflect the structure you are using before testing its operation.

.....

## CreateParentZone

```
#!/bin/env adedit
# This script creates a tree zone. Use this, for example,
to create the
# parent zone for child zones created in other scripts.
package require ade_lib
proc usage {msg} {
    puts {usage: -z >parentZone> -u >user>}
    puts $msg
    exit 1
}
if {[getopt argv -z parentZone] == 0} {
    usage "Missing the name for the new zone"
}
puts "
Enter the domain name for the bind command"
gets stdin domain
if {[getopt argv -u user] != 0} {
    if {[getopt argv -p password]} {
        bind $domain $user $password
    } else {
        bind $domain $user
    } else {
        puts "Enter administrator name"
        gets stdin user
        bind $domain $user
    }
}
set domaindn [dn_from_domain $domain]
puts "
Enter the name of the Active Directory container that holds
the Centrify zone data"
gets stdin zonesNode
puts "
Enter the organizational unit with the Centrify zone data
container"
gets stdin baseOU
puts "Summary:"
puts "Domain is $domain. DN for the domain is $domaindn"
puts "The base OU is $baseOU."
puts "The container for the zone information is
$zonesNode"
puts "The new zone is named $parentZone"
#create the parent zone in Active Directory
puts "
```

.....

```
Creating Centrify zone $parentZone"
create_zone tree
"cn=$parentZone,cn=$zonesNode,ou=$baseOU,$domaindn" std
puts "Created new zone:
cn=$parentZone,cn=$zonesNode,ou=$baseOU,$domaindn"
```

## Create child zones

This script creates two child zones in the domain and parent zone specified in the command line. The command line syntax is as follows:

```
>./CreateChildZones -d domain -z parentZone [-u adminName]
[-p password]
```

where:

- *domain* is the domain name
- *parentZone* is the name of an existing zone
- *adminName* is the name of an Active Directory user with administrator privileges on the domain controller
- *password* is the administrator's password. If you do not enter the password in the command line, you are prompted for it

The *password* is optional. If you do not type it in the command line, the script prompts you to enter it.

The script binds to the domain you specify using the user name and password you provide. The script then prompts you to enter the name of the organizational unit and container in which you store the zone information. After that, it prompts you to enter names for the two child zones. Note that this sample script assumes you are using the default deployment structure with the top-level organizational unit. If you are not using the default deployment structure, you should modify the sample script to reflect the structure you are using before testing its operation.

To confirm the script ran successfully, open Access Manager and expand the **Child Zones** node under the parent zone you specified in the command line. If the two new child zones are listed, you can right-click each zone name to see its zone properties.

.....

## CreateChildZones

```
#!/bin/env adedit
# This script creates 2 child zones in the domain and
parent zone
# specified in the command line.
#
package require ade_lib
proc usage {msg} {
    puts {usage: -d <domain> -z <parentZone> [-u <user>] [-p
<password>]}
    puts $msg
    exit 1
}
if {[getopt argv -d domain] == 0} {
    usage "Missing Domain, ex. demo.test"
}
if {[getopt argv -z parentZone] == 0} {
    usage "Missing parent zone, ex. HQ"
}
if {[getopt argv -u user] != 0} {
    if {[getopt argv -p password]} {
        bind $domain $user $password
    } else {
        bind $domain $user
    } else {
        puts "Enter administrator name"
        gets stdin user
        bind $domain $user
    }
}
puts "
Enter the name of the container for the Centrify zone data"
gets stdin zoneContainer
puts "
Enter the organizational unit for the Centrify zone data"
gets stdin zoneContainerOU
# Define distinguished name for domain
set domaindn [dn_from_domain $domain]
puts "
Summary:"
puts "Domain is $domain. DN for the domain is $domaindn"
puts "The base OU is $zoneContainerOU."
puts "The container for the zone information is
$zoneContainer
"
```

.....

```
# Create child zones
puts "Enter child zone name"
gets stdin czone1
puts "
Enter another child zone name"
gets stdin czone2
create_zone tree
"cn=$czone1,cn=$parentZone,cn=$zoneContainer,ou=$zoneContai
nerOU,$domaindn" std
create_zone tree
"cn=$czone2,cn=$parentZone,cn=$zoneContainer,ou=$zoneContai
nerOU,$domaindn" std
# link the children to parent
select_zone
"cn=$czone1,cn=$parentZone,cn=$zoneContainer,ou=$zoneContai
nerOU,$domaindn"
set_zone_field parent
"cn=$parentZone,cn=$zoneContainer,ou=$zoneContainerOU,$doma
indn"
save_zone
select_zone
"cn=$czone2,cn=$parentZone,cn=$zoneContainer,ou=$zoneContai
nerOU,$domaindn"
set_zone_field parent
"cn=$parentZone,cn=$zoneContainer,ou=$zoneContainerOU,$doma
indn"
save_zone
puts "
Child zones $czone1 and $czone2 created in $parentZone"
```

## Create privileged commands and roles

Users get the rights necessary to run privileged commands and access applications from their role assignments. The predefined UNIX Login role gives users basic access to UNIX computers without any elevated privileges. The scripts in this section illustrate how you can create roles with additional rights. The first sample script uses a separate text file to define a new role and the commands users in that role are allowed to execute. The second sample script illustrates how to define the commands and the role within the script after prompting for bind credentials and the target zone.

Both scripts create the same commands and role.



## Privileges and role defined in a file

For the first sample script, a single role and its privileged commands are defined in the file `Role_apacheAdmin.txt`. This sample text file defines the role name and a few sample commands that you might assign to an Apache server administrator. For example:

```
ApacheAdminRole
vi /etc/httpd/conf/httpd.conf
apachectl *
htpasswd *
```

The first line in the `Role_apacheAdmin.txt` file specifies the new role name. The subsequent lines specify the commands to add to the role. You can edit the text file to suit your environment. For example, you might want add or remove commands or modify the path to the Apache configuration file. To create the role and commands, you can then run the `MakeRole` sample script and specify the `Role_apacheAdmin.txt` file name as a command-line argument. The `MakeRole` sample script then prompts you to enter the domain name, account, and password for the `bind` command and to type the name of the parent zone where the sample role will be created.

Note that you must specify a parent zone for this sample script. The second sample `ApacheAdminRole` script shown in [Privileges and roles defined in the script](#) displays the list of zones in the domain to illustrate how you can create a role in a child zone. In addition, this sample script assumes you are using the default deployment structure with the top-level organizational unit. If you are not using the default deployment structure, you should modify the sample script to reflect the structure you are using before testing its operation.

## MakeRole

The `MakeRole` sample script creates a role with the set of privileged commands defined in the sample `Role_apacheAdmin.txt` file.

```
#!/bin/env adedit
# This script creates a role consisting of a
# set of privileged commands
# The role name and commands are specified
# in a separate file.
#
# The first line in the input file should be
# the new role name.
```

.....

```
# The subsequent lines are the names of the
# privileged commands to
# add to the role.
# For example:
#     audit_admin_cmds
#     /usr/bin/vi /etc/security/audit/config
#     /usr/bin/vi /etc/security/audit/objects
package require ade_lib
if { $argc != 1 } {
    puts "usage: $argv0 file"
    exit 1
}
if {[catch {set fp [open [lindex $argv 0] r]} errmsg]}
{
    puts "Cannot open [lindex $argv 0]."
    exit 1
}
# Get domain and bind
puts "Enter domain name"
gets stdin domain
set domaindn [dn_from_domain $domain]
puts "Enter account name with administrator privileges"
gets stdin administrator
puts "Enter $administrator password"
gets stdin APWD
bind $domain $administrator "$APWD"
# Select the target zone and base organizational unit
puts "Enter the target zone name for the new role"
gets stdin zonename
puts "
Enter the name of the Active Directory
    container that holds the Centrify zone data"
gets stdin zonesNode
puts "
Enter the organizational unit with the Centrify zone data
container"
gets stdin baseOU
select_zone
"cn=$zonename,cn=$zonesNode,ou=$baseOU,$domaindn"
if {[gets $fp line] == -1} {
    puts "Cannot read [lindex $argv 0]."
    exit 1
}
# Create role
puts "Creating role...$line"
```

.....

```
set role $line
new_role "$role"
save_role "$role"
set count 0
while {[gets $fp line] >= 0} {
    incr count
    # Create command. Each command will be named
    # based on the role defined in the first line
    # and the command's line number in the file
    set cmd_name $role$count
    new_dz_command "$cmd_name"
    # set the command fields
    set cmd_path $line
    set_dzc_field cmd "$cmd_path"
    set_dzc_field dzdo_runas root
    set_dzc_field umask 077
    # prevent nested execution
    set_dzc_field flags 1
    # save the command
    save_dz_command
    # Add the command to the Role
    add_command_to_role "$cmd_name"
}
close $fp
save_role "$role"
```

## Privileges and roles defined in the script

In this sample script, you create the same Apache administrator commands and role as the previous script. However, this script displays a list of the zones in the domain and lets you select in which zone to create the commands and role.

### ApacheAdminRole

```
#!/bin/env adedit
puts "This script creates privileged commands and the
ApacheAdminRole in the zone entered"
package require ade_lib
puts "
Enter the domain name"
gets stdin domain
puts "
Enter the account name to use to modify Active Directory"
```

.....

```
gets stdin acctName
bind $domain $acctName
set domaindn [dn_from_domain $domain]
set zonelist [get_zones $domain]
set numberZones [llength $zonelist]
set row 1
set zonenum 0
puts "
This domain contains the following zones"
while {$numberZones != 0} {
    puts "$row. [lindex $zonelist $zonenum]"
    incr zonenum
    incr row
    incr numberZones -1
}
puts "
Enter the row number of the target zone"
gets stdin rowSelect
set zone [lindex $zonelist [incr rowSelect -1]]
select_zone "$zone"
puts "
Creating command-level Apache admin rights in $zone"
puts "
Creating web_edit_httpd_config"
new_dz_command web_edit_httpd_config
set_dzc_field cmd "vi /etc/httpd/conf/httpd.conf"
set_dzc_field description "edit httpd config file"
set_dzc_field dzdo_runas root
set_dzc_field dzsh_runas root
set_dzc_field path /usr/local/apache2/bin
set_dzc_field flags 1
save_dz_command
puts "
Creating web_apachectl"
new_dz_command web_apachectl
set_dzc_field cmd "apachectl *"
set_dzc_field description "Web Apache Server Control"
set_dzc_field dzdo_runas root
set_dzc_field dzsh_runas root
set_dzc_field path /usr/local/apache2/bin
save_dz_command
puts "
Creating web_htpasswd"
new_dz_command web_htpasswd
set_dzc_field cmd "htpasswd *"
```

.....

```
set_dzc_field description "Web Apache Manage user files"
set_dzc_field dzdo_runas root
set_dzc_field dzsh_runas root
set_dzc_field path /usr/local/apache2/bin
save_dz_command
#-----
-----
# Create ApachedAdminRights role
# The new_role command creates the role in the currently
selected zone.
puts "
Creating the ApacheAdminRole with these rights"
# In each role you need to set the sysrights with the set_
role_field
# to the following binary values
# password_login = 01
# sso = 02
# ignore_disabled = 04
# full_shell = 08
new_role ApacheAdminRights
add_command_to_role web_edit_httpd_config
add_command_to_role web_apachectl
add_command_to_role web_htpasswd
set_role_field sysrights [expr 0x0000000b] #full_shell |
sso | password_login
save_role
save_zone
```

## Add and provision UNIX users

It is difficult to provision a lot of UNIX users and ensure that the UID is unique in the domain. To assist you with the process, Centrify provides a set of features called the Zone Provisioning Agent. The Zone Provisioning Agent includes a service that automatically assigns a unique UID and other UNIX profile attributes, such as the home directory, default shell, and primary GID, based on rules you define.

This script demonstrates how you could use the Zone Provisioning Agent to add and provision users. For this sample script, the list of UNIX users is defined in the source file named `users.txt` and the Active Directory source group is `unix Users`.



**Note** To learn more about the Zone Provisioning Agent and automated provisioning, see the *Planning and Deployment Guide*.

## users.txt

You specify the names to be added in a text file in which each name is on a separate line. Be sure to use line feed only as the end-of-line; do not use CR-LF. The sample file in the distribution package contains the following names:

```
Amy.Adams  
Brenda.Butler  
Dennis.Day  
Eric.Edwards
```

## AddUnixUsers

In the following script, you specify the file name with the user names in the command line. The script then prompts you for the additional information required. The target Active Directory group—`unix users`—is hard-coded into the script.

This script uses the Tcl `catch` command three times to control processing when an error occurs.

- In the first case, it is used to exit gracefully if the specified file cannot be opened.
- In the second case, `catch` is used to determine if the user already exists. An error here indicates that the user does not exist and, rather than exiting, the `else` statement creates the user. (If the user already existed, you would not want to create another Active Directory account.)
- In the third case, `catch` is used to exit gracefully if the user is already a member of the `unix users` group.

```
#!/bin/env adedit  
# This script creates an Active Directory account  
# for each user the specified  
# and adds the user to UNIX Users group.  
# This automatically fills in their UNIX profile.  
# Command line input: file name w/ user names in
```

.....

```
# format ffff.llll only
# Prompted input: domain, administrator
#name, default password
package require ade_lib
if { $argc != 1 } {
    puts "usage: $argv0 file"
    exit 1
}
if {[catch {set users [open [lindex $argv 0] r]}
    errmsg]} {
    puts "Cannot open [lindex $argv 0]."
    exit 1
}
# Get domain and bind
puts "Enter domain name"
gets stdin domain
set domaindn [dn_from_domain $domain]
puts "Enter account name with administrator privileges"
gets stdin administrator
puts "Enter $administrator password"
gets stdin APWD
bind $domain $administrator "$APWD"
puts "
Define password to be used for all accounts"
gets stdin pwd
# Now start creating accounts from users
# example: "cn=Ellen Edwards,cn=Users,$domaindn"
# "Ellen.Edwards@$domain" ellen.edwards pwd
while {[gets $users sam] >= 0} {
    set name [split $sam .]
    set dn "cn=[lindex $name 0] [lindex $name 1],
        cn=Users,$domaindn"
    set upn $sam@$domain
    if { [catch { select_object $dn }] } {
        # If we fail to select the object,
        # most probably it
        # does not exist. So we create it here.
        puts "Creating $dn"
        create_aduser $dn $upn $sam $pwd
    } else {
        puts "$dn exists. Skip creating."
    }
}
# Because we already installed and started ZPA,
# this provisions the
# Active Directory account
```

.....

```
        catch { add_user_to_group $sam@$domain
                "UNIX Users@$domain" }
    }
close $users
```

## Simple tools

The following scripts are simple “utilities” for getting information from Active Directory about the managed computers and users accounts:

- **computers-report**: Lists the managed computers in the zone.
- **useracc-report**: List the Active Directory users in the domain and several account properties.
- **user-report**: Lists the users in a zone.
- **GetComputers**: Lists all of the managed computers in the specified domain and the zone to which each computer is joined.

Following these scripts are sample scripts that demonstrate how you can use a script that calls, for example, commonly-used commands in other scripts. For more information, see [Run a script from a script](#).

### computers-report

Use this command to list managed computers in the zone. The command line arguments are as follows:

Label	Required/Optional	Description
-domain	required	Domain name.
		Bind using the ADEdit host computer’s credentials (see <a href="#">bind</a> ).
-m	optional	You can use either the computer credentials (-m) or the user account credentials (-u).
-u	optional	Administrator’s account name.
		Administrator’s account password.
-p	optional	<b>Note:</b> If you do not enter the password in the command line you will be prompted to enter it.
-sep	optional	Separator used between data. The default is separator is the pipe ( ) character.

.....

```
#!/bin/env adedit
# This script lists the managed computers on the zone.
# Command line input is the domain, the
# administrator account name and
# the separator to use between computer's field
# values in the output
package require ade_lib
# Lists all of the managed computers and the zone
proc usage {msg} {
    puts {usage: -domain <domain> [-m] [-u <user>]
        [-p <password>] [-sep csv | tab | <char>]}
    puts $msg
    exit 1
}
if {[getopt argv -domain domain] == 0} {
    usage "Missing domain"
}
set verbose 0
if {[getopt argv -v]} {
    set verbose 1
}
set sep "|"
getopt argv -sep sep
if {$sep == "csv"} {set sep ","}
if {$sep == "tab"} {set sep "\t"}
if {[getopt argv -m]} {
    bind -gc -machine $domain
} else {
    if {[getopt argv -u user]} {
        if {[getopt argv -p password]} {
            bind -gc $domain $user $password
        } else {
            bind -gc $domain $user
        }
    } else {
        bind -gc $domain
    }
}
# this code runs entirely off the GC
cache on
set scps [get_objects -gc -depth sub [dn_from_domain
$domain] {(&(displayName=$CimsComputerVersion*)
(objectClass=serviceConnectionPoint))}]
foreach scp $scps {
    select_object -gc $scp
    set name [get_object_field name]
```

.....

```
set parent ""
# first look for parentLink
foreach k [get_object_field keywords] {
  set bits [split $k ':']
  if {[lindex $bits 0] == "parentLink"} {
    set sid [lindex $bits 1]
    #ok we got it
    # make sure it exists
    catch {set parent [principal_from_sid $sid]}
  }
}
# if we didn't then try by managed By (DC3)
if {$parent == ""} {
  set mb [get_object_field managedBy]
  if {$mb != ""} {
    set parent $mb
  }
}
set orphan 0
if {$parent == ""} {set orphan 1}
set path [get_parent_dn [get_parent_dn
  [get_object_field dn]]]
set zone [string range [get_rdn $path] 3 end]
puts $name$sep$zone$sep$orphan
}
```

## useracc-report

Use this command to list all users and their Active Directory account control values. The command line arguments are as follows:

Label	Required/Optional	Description
-domain	required	Domain name
-m	optional	Bind using the ADEdit host machine's credentials (see <a href="#">bind</a> ) Note: If you use -m you do not need to enter -u
-u	optional	Administrator's account name.
-p	optional	Administrator's account password. Note: If you do not enter the password in the command line you will be prompted to enter it.
-sep	optional	Separator used between data. Default is

.....

```
#!/bin/env adedit
# This script lists all the users and their Active
Directory account control values
package require ade_lib
# List users and the following field
proc usage {msg} {
    puts {usage: -domain <domain> [-m] [-u
<user>] [-p <password>] [-sep csv | tab | <char>]}
    puts $msg
    exit 1
}
if {[getopt argv -domain domain] == 0} {
    usage "Missing domain"
}
set verbose 0
if {[getopt argv -v]} {
    set verbose 1
}
set sep "|"
getopt argv -sep sep
if {$sep == "csv"} {set sep ","}
if {$sep == "tab"} {set sep "\t"}
if {[getopt argv -m]} {
    bind -machine $domain
} else {
    if {[getopt argv -u user]} {
        if {[getopt argv -p password]} {
            bind $domain $user $password
        } else {
            bind $domain $user}
        } else {
            bind $domain
        }
    }
}
cache on
proc my_convert_msdate {msdate} {
    if {$msdate==9223372036854775807} {
        return -1
    }
    return [clock format [expr ($msdate/10000000)-
11644473600] -format "%m/%d/%y %H:%M:%S"]
}
proc nice_date {date} {
    if {$date == ""} {return $date}
    if {$date == 0} {return ""}
```

.....

```
set ret [my_convert_msdate $date]
if {$ret == -1} {return ""}
return $ret;
}
set users [get_objects -depth sub [dn_from_domain $domain]
"(objectcategory=Person)"]
foreach user $users {
    select_object $user
    set uac [get_object_field userAccountControl]
    if {$uac == ""} {continue}
    # gof is get_object_field
    eval "set name [gof cn]"
    #puts [gof dn]
    set sam [gof sAMAccountName]
    set exp [nice_date [gof accountExpires] ]
    set locked [nice_date [gof lockoutTime] ]
    set lastlogon [nice_date [gof lastLogon] ]
    set enabled [expr $uac&0x2 ]
    set enabstr "False"
    if {$enabled} {set enabstr "True"}
    puts
$name$sep$sam$sep$exp$sep$locked$sep$lastlogon$sep$enabstr
}
```

## user-report

Use this command to lists the users in the specified zone. The command line arguments are as follows:

Label	Required/Optional	Description
-z	required	The distinguished name of the zone
-m	optional	Bind using the ADEdit host machine's credentials (see <a href="#">bind</a> ) Note: If you use -m you do not need to enter -u
-u	optional	Administrator's account name. Administrator's account password.
-p	optional	Note: If you do not enter the password in the command line you will be prompted to enter it.

```
#!/bin/env adedit
# This script lists the users in the zone you specify in
the command line.
# On the command line use either -m or -u
```

.....

```
package require ade_lib
proc usage {msg} {
    puts {usage: -z <zoneDN> [-m] [-u <user>] [-p
<password>]}
    puts $msg
    exit 1
}
if {[getopt argv -z zoneDN] == 0} {
    usage "Missing input zone. Enter full
distinguished name"
}
if {[catch {domain_from_dn $zoneDN} domain]} {
    usage "Invalid input zone name. Enter full
distinguished name"
}
set verbose 0
if {[getopt argv -v]} {
    set verbose 1
}
if {[getopt argv -m]} {
    bind -machine $domain
} else {
    if {[getopt argv -u user]} {
        if {[getopt argv -p password]} {
            bind $domain $user $password
        } else {
            bind $domain $user
        } else {
            bind $domain
        }
    }
}
select_zone $zoneDN
list_zone_users
```

## GetComputers

Use this command to list all the Centrify-managed computers in the specified domain. Enter the domain name in the command line.

```
#!/bin/env adedit
# GetComputers
# Purpose: Retrieves a listing of all UNIX computers in all
Centrify Zones.
package require ade_lib
```

.....

```
puts "  
This script retrieves a listing of all UNIX computers in  
the specified domain"  
puts "and shows the zone to which it is joined"  
if { $argc == 0 } {  
    puts "  
    Command format: $argv0 domain name"  
    exit 1  
}  
set domain [lindex $argv 0]  
# Use lindex command because argv is a list and bind  
requires a string  
puts "  
Enter administrator name for bind command"  
gets stdin admin  
bind $domain $admin  
foreach ZONE [get_zones $domain] {  
    select_zone $ZONE  
    foreach COMPUTER [get_zone_computers] {  
        puts -nonewline $COMPUTER::  
puts $ZONE;  
    }  
}
```

## Run a script from a script

The following scripts illustrate the use of the Tcl source command to run the script in a specified file. In this example, the source file is `setenv`, which prompts the user to enter environment variables such as the domain and zone.

**Note** You may find repeated use of `setenv` to be maddening since it prompts you for all of the environment variables regardless of whether the command actually needs them. This is done for demonstration purposes only. In a production environment, you would eliminate the prompts you don't need by tailoring `setenv` specifically to your environment. Feel free to remove or comment out parts when you've had enough.

The subsequent scripts in this section call the `setenv` script and then run a short script that does simple queries, such as get the child zones, get the computers in the zone, and get the groups.

.....

## setenv

This script prompts you to enter data that can be used in the calling script. This example is intended as a demonstration only. Not all of the information is relevant to the calling script. Note that this sample script assumes you are using the default deployment structure with a top-level organizational unit. If you are not using the default deployment structure, you should modify the sample script to reflect the structure you are using before testing its operation.

```
# Setenv file contents
# Purpose: Sets up a common environment for the following
Active Directory
# tools, selecting the Active Directory Domain, binding the
user, and
# defining commonly used variables.
# Other Active Directory tools:
# GetZones
# GetUsers
# GetGroups
# GetChildZones
# GetComputers
puts "
This portion of the script prompts you to enter the domain
and account name for the bind command."
# If you are always using the same domain, comment out the
puts and gets and use the set command instead
puts "
Enter the domain name"
gets stdin domain
# get the distinguished name for the domain.
set domaindn [dn_from_domain $domain]
puts "
Enter administrator account name for bind command"
gets stdin admin
bind $domain $admin
puts "
bind to $domain complete"
puts "
The next two prompts ask you to enter the OU and container
for your zone information"
puts "
Enter the name of the Active Directory container that holds
the Centrify zone-related data"
```

.....

```
gets stdin zonesContainer
# If you are always using the same zone, comment out the
puts and gets and use the set command instead
# set zonesContainer <Active Directory container with zones
data>
puts "
Enter the name of the organizational unit that has the zone
container."
gets stdin zonesContainerOU
# If you are always using the same OU for the zone
container, comment out the puts and gets and use the set
command instead
# set zonesContainerOU <Active Directory OU with zones
container>
puts "
Enter the base organizational unit with the Centrify
managed computers data"
gets stdin baseOU
# If you are always using the same base OU, comment out the
puts and gets commands and use the set command instead
# set baseOU <base OU name>
puts "
The next prompt asks for the parent zone."
# If you are always using the same zone, comment out the
puts and gets and use the set command instead
# set parentZone <parent zone name>
puts "
Enter the parent zone name"
gets stdin parentZone
```

## GetZones

Use this script to get a list of all the zones in a domain.

```
#!/bin/env adedit
# GetZones
# Purpose: Performs a recursive listing of all Centrify
zones in the specified
# domain
package require ade_lib
source setenv
puts "
This script retrieves a recursive listing of all Centrify
zones in the $domain domain"
```

.....

```
puts "  
The Active Directory folder with the Centrify zone data is  
named $zonesContainer"  
puts "  
That container is in organizational unit $zonesContainerOU"  
puts "  
The parent zone is $parentZone"  
foreach ZONE [get_zones $domain] {  
    puts $ZONE;  
}
```

## GetUsers

Use this script to get a list of all users in a zone.

```
#!/bin/env adedit  
# GetUsers  
# Purpose: Operates on a recursive listing of all UNIX  
users in all  
# Centrify Zones, and retrieves the administered UNIX  
attribute values  
# for each user object in each zone.  
package require ade_lib  
puts "  
This script retrieves the UNIX attributes for each user in  
each zone in the specified domain"  
source setenv  
foreach ZONE [get_zones $domain] {  
    select_zone $ZONE  
    foreach USER [get_zone_users] {  
        save_zone_user $USER  
        puts -nonewline "[get_zone_user_field uname]:  
[gzuf uid]:[gzuf gid]:[gzuf gecos]:[gzuf home]:[gzuf  
shell]"; puts :$USER:$ZONE  
    }  
}
```

## GetGroups

Use this script to get the UNIX group attribute values for the groups in the managed computers.

.....

```
#!/usr/bin/env adedit
# GetGroups
# Purpose: Retrieves the UNIX group attribute values for
each UNIX
# group administered in the parent zone specified in
setenv.
# To select a different zone, change the DN in the select_
zone command
package require ade_lib
puts "
This script retrieves the group attribute values for each
UNIX group in the specified parent zone"
source setenv
select_zone
"CN=$parentZone,CN=$zonesContainer,OU=$zonesContainerOU,$do
maindn"
foreach GROUP [get_zone_groups] {
    select_zone_group $GROUP
    puts -nonewline "[get_zone_group_field name]:[gzgf
gid]"; puts :$GROUP
}
```

## GetChildZones

Use this command to get a list of the child zones for the specified parent.

```
#!/bin/env adedit
# # GetChildZones
# Purpose: Retrieves a recursive listing of all new
hierarchical Centrify child
# zones administered underneath the parent zone specified
in setenv
#
package require ade_lib
source setenv
puts "
This script retrieves a recursive listing of all child
zones in $parentZone"
puts "
The Active Directory folder with the Centrify zone
information is $zonesContainer"
select_zone
"CN=$parentZone,CN=$zonesContainer,OU=$zonesContainerOU,$do
maindn"
```

.....

```
foreach ZONE [get_child_zones -tree] {  
    puts $ZONE;  
}
```

# ADEdit command reference

This chapter describes each ADEdit command in alphabetical order. Each command description includes details about the options and arguments you can specify and the values returned, if applicable.

In addition, some ADEdit commands can only be used when you are working with hierarchical zones. Other commands can be used in classic or hierarchical zones, but require you to specify the zone type. For each command, the **Zone type** section indicates whether there are any zone-related constraints as follows:

- **Hierarchical only:** You must have a hierarchical zone selected for the command to work.
- **Classic and hierarchical:** You can use the command in both classic zones and hierarchical zones. Options in the command let you specify whether you are working with a classic or hierarchical zone. In most cases, commands that work in both classic and hierarchical zones, require the classic zone to be a classic4 zone. The classic3 zone type is intended for backward compatibility with older agents and only commands where the zone type is not applicable are supported.
- **Classic only:** You must have a classic4 zone selected for the command to work.
- **Not applicable:** You can use the command because the zone type does not matter.

In addition to the zone type, syntax, and return values, each command description includes at least one usage example and a summary of related commands, if appropriate.



## add\_command\_to\_role

Use the `add_command_to_role` command to add a privileged UNIX command to the currently selected role that is stored in memory. The command must already exist. You can create privileged UNIX commands using `new_dz_command`.

The `add_command_to_role` command does *not* change the role as it is stored Active Directory. Running the command changes the role only in memory. You must save the role before the added command takes effect in Active Directory. If you select another role or quit ADEdit before saving the role, any commands you've added since the last save won't take effect.

## Zone type

Classic and hierarchical

## Syntax

```
add_command_to_role command[/zonename]
```

## Abbreviation

acr

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>command[ /zonename ]</code>	string	Required. Specifies the name of an existing UNIX command to add to the currently selected role.  If the UNIX command right that you want to add is defined in the current zone, the <i>zonename</i> argument is optional. If the UNIX command right is defined in a zone other than the currently selected zone, the <i>zonename</i> argument is required to identify the specific UNIX command right to add.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
add_command_to_role basicshell/global
```

This example adds the command `basicshell`, defined in the `global` zone, to the currently selected role.

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select a role to work with:

- `get_role_commands` returns a Tcl list of the UNIX commands for the role.
- `new_role` creates a new role.
- `select_role` retrieves a role from Active Directory.

The following commands enable you to work with a currently selected role:

- `add_pamapp_to_role` adds a PAM application to the role.
- `delete_role` deletes the selected role from Active Directory and from memory.
- `get_role_apps` returns a Tcl list of the PAM applications for the role.
- `get_role_field` reads a field value from the role.



- `list_role_rights` lists of all privileged commands and PAM application rights for the role.
- `remove_command_from_role` removes a UNIX command from the role.
- `remove_pamapp_from_role` removes a PAM application from the role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the role.

## add\_map\_entry

Use the `add_map_entry` command to add an entry to the currently selected NIS map stored in memory. This command does not support a comment field. If you want to add a comment along with the entry use `add_map_entry_with_comment` instead.

To change an existing entry in a NIS map, use `delete_map_entry` to remove the entry, then add the revised version using `add_map_entry`.

The `add_map_entry` command changes the NIS map in memory and in Active Directory. You do not need to save the NIS map for the added entry to take effect in Active Directory.

## Zone type

Not applicable

## Syntax

```
add_map_entry key value
```

## Abbreviation

ame



## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
key	string	Required. Specifies the key of the NIS map entry.
value	string	Required. Specifies the value of the NIS map entry.

## Return value

This command returns nothing if it runs successfully.

## Example

```
add_map_entry Finance "Hank@acme.com,Sue@acme.com"
```

This example adds the NIS map entry `Finance` with the value `Hank@acme.com,sue@acme.com` to the currently selected NIS map.

## Related commands

The following commands enable you to view and select the NIS map you want to work with:

- `get_nis_maps` returns a Tcl list of NIS maps in the currently selected zone.
- `list_nis_maps` lists to `stdout` of all NIS maps in the currently selected zone.
- `new_nis_map` creates a new NIS map and stores it in memory.



- `select_nis_map` retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use additional commands to work with that map's entries or use the following commands to delete or save the currently selected NIS map:

- `delete_nis_map` deletes the selected NIS map from Active Directory and from memory.
- `save_nis_map` saves the selected NIS map with its current entries to Active Directory.

## add\_map\_entry\_with\_comment

Use the `add_map_entry_with_comment` command to add an entry to the currently selected NIS map stored in memory and lets you include a comment. The comment can be up to 2048 characters and does not support new line syntax.

To change an existing entry in a NIS map, use `delete_map_entry` to remove the entry, then add the revised version using `add_map_entry_with_comment`.

The `add_map_entry_with_comment` command changes the NIS map in memory and in Active Directory. You do not need to save the NIS map for the added entry to take effect in Active Directory.

## Zone type

Not applicable

## Syntax

```
add_map_entry_with_comment key value comment
```

## Abbreviation

amewc

.....

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
key	string	Required. Specifies the key of the NIS map entry.
value	string	Required. Specifies the value of the NIS map entry.
comment	string	Required. Specifies the comment for the NIS map entry.

## Return value

This command returns nothing if it runs successfully.

## Example

```
add_map_entry_with_comment Finance  
"Hank@acme.com,Sue@acme.com" "new Finance staff"
```

This example adds the NIS map entry `Finance`, with the value `Hank@acme.com,Sue@acme.com` and comment `new Finance staff` to the currently selected NIS map.

## Related commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and select a NIS map to work with:

- `get_nis_maps` returns a Tcl list of NIS maps in the current zone.
- `list_nis_maps` lists to `stdout` the NIS maps in the current zone.



- `new_nis_map` creates a new NIS map.
- `select_nis_map` retrieves a NIS map from Active Directory.

The following commands enable you to work with a currently selected NIS map:

- `add_map_entry` adds an entry to the NIS map.
- `delete_map_entry` removes an entry from the NIS map.
- `get_nis_map_field` reads a field value from the NIS map.
- `get_nis_map` and `get_nis_map_with_comment` return a Tcl list of NIS map entries.
- `list_nis_map` and `list_nis_map_with_comment` lists NIS map entries to `stdout`.

## add\_object\_value

Use the `add_object_value` command to add a value to a multi-valued field (attribute) of a specified Active Directory object in Active Directory. This command only works on the object in Active Directory, not on the currently selected Active Directory object in memory (if there is one).

If the added value isn't valid, Active Directory will report an error and `add_object_value` won't save the value.

This command is useful for fields that may be very large—members of a group, for example.

## Zone type

Not applicable

## Syntax

```
add_object_value dn field value
```



## Abbreviation

aov

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
dn	string	Required. Specifies the distinguished name (DN) of the Active Directory object in which to add a value.
field	string	Required. Specifies the name of a multi-valued field in the currently selected Active Directory object to which to add the value. This can be any field that is valid for the type of the currently selected Active Directory object.
value		Required. Specifies the value to add to the field. The type of value depends on the field specified by the <i>field</i> argument.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
add_object_value cn=groups,dc=acme,dc=com users adam.avery
```

This example adds the value `adam.avery` to the `users` field of the `groups` object specified by the DN.

## Related commands

The following commands enable you to work with Active Directory objects:



- `delete_object` deletes the Active Directory object from Active Directory.
- `delete_sub_tree` deletes the Active Directory object and all of its children.
- `get_object_field` reads a field value from the Active Directory object.
- `remove_object_value` removes a value from a multi-valued attribute of the Active Directory object.
- `save_object` saves the Active Directory object.
- `set_object_field` sets a field value in the Active Directory object.

## add\_pamapp\_to\_role

Use the `add_pamapp_to_role` command to add a PAM application right to the currently selected role stored in memory. The PAM application right must already exist. You can create PAM application rights using `new_pam_app`.

The `add_pamapp_to_role` command does *not* change the role as it is stored Active Directory. The command only changes the role stored in memory. You must save the role using `save_role` before the added PAM application takes effect in Active Directory. If you select another role or quit ADEdit before saving the role, any PAM application rights you've added since the last save won't take effect.

You can only use the `add_pamapp_to_role` if the currently selected zone is a classic4 or hierarchical zones. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
add_pamapp_to_role app[/zonename]
```

.....

## Abbreviation

`apr`

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>app[ /zonename ]</code>	string	Required. Specifies the name of an existing PAM application right to add to the currently selected role.  If the PAM application right that you want to add is defined in the current zone, the <i>zonename</i> argument is optional. If the PAM application right is defined in a zone other than the currently selected zone, the <i>zonename</i> argument is required to identify the specific PAM application right to add.

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example adds the PAM application `login-all`, which is defined in the currently selected zone, to the currently selected role:

```
add_pamapp_to_role login-all
```

The following example adds the PAM application access right `oracle-admin` from the `emea` zone to the currently selected role:

```
add_pamapp_to_role oracle-admin/emea
```



## Related commands

The following commands enable you to view and select the role you want to work with:

- `new_role` creates a new role and stores it in memory.
- `select_role` retrieves a role from Active Directory and stores it in memory.
- `get_roles` returns a Tcl list of roles in the current zone.
- `list_roles` displays a list to stdout of all roles in the currently selected zone.

After you have a role stored in memory, you can use additional commands to work with that role's fields, commands, and applications or use the following commands to delete or save the currently selected role:

- `save_role` saves the selected role with its current settings to Active Directory.
- `delete_role` deletes the selected role from Active Directory and from memory.

## add\_sd\_ace

Use the `add_sd_ace` command to add an access control entry (ACE) in ACE string form to a security descriptor (SD) in SDDL (security descriptor description language) form.

The command takes an ACE string and an SDDL string. The command writes the ACE string there. The command returns an SDDL string that includes the added ACE string.

## Zone type

Not applicable

.....

## Syntax

```
add_sd_ace sddl_string ace_string
```

## Abbreviation

ase

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
sddl_string	string	Required. Specifies a security descriptor in SDDL format.
ace_string	string	Required. Specifies an access control entry in ACE string form (which is always enclosed in parentheses)

## Return value

This command returns a security descriptor string in SDDL format if it runs successfully.

## Examples

This example adds an ACE string to an SDDL. The ACE string to add is at the end of the command in **boldface**:

```
add_sd_ace O:DAG:DAD:AI(A;;;RCWDWOCDCCLCSWRPWPLOCR;;;DA)
(OA;;;CCDC;bf967aba-0de6-11d0-a285-00aa003049e2;;;AO)
```

.....

```
(OA;;CCDC;bf967a9c-0de6-11d0-a285-00aa003049e2;;AO)
(OA;;CCDC;bf967aa8-0de6-11d0-a285-00aa003049e2;;PO)
(A;;RCLCRPLO;;;AU) (OA;;CCDC;4828cc14-1437-45bc-9b07-
ad6f015e5f28;;AO)
(OA;CIIOID;RP;4c164200-20c0-11d0-a768-
00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;4c164200-20c0-11d0-a768-
00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a86-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a9c-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967aba-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RCLCRPLO;;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RCLCRPLO;;bf967a9c-
0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RCLCRPLO;;bf967aba-0de6-11d0-a285-
00aa003049e2;RU)
(OA;CIID;RPWPCR;91e647de-d96f-4b70-9557-d63ff4f3ccd8;;PS)
(A;CIID;SDRCWDWOCCDCLCSWRPWPDTLOCR;;;EA)
(A;CIID;LC;;;RU) (A;CIID;SDRCWDWOCCLCSWRPWPLOCR;;;BA)
(A;;SDRCWDWOCCDCLCSWRPWPDTLOCR;;;SY)
```

This example returns:

```
O:DAG:DAD:AI (A;;SDRCWDWOCCDCLCSWRPWPDTLOCR;;;SY)
(A;;RCWDWOCCDCLCSWRPWPLOCR;;;DA)
(OA;;CCDC;bf967aba-0de6-11d0-a285-00aa003049e2;;AO)
(OA;;CCDC;bf967a9c-0de6-11d0-a285-00aa003049e2;;AO)
```

.....

```
(OA;;CCDC;bf967aa8-0de6-11d0-a285-00aa003049e2;;PO)
(A;;RCLCRPLO;;;AU) (OA;;CCDC;4828cc14-1437-45bc-
9b07-ad6f015e5f28;;AO) (OA;CIIOID;RP;4c164200-20c0-11d0-
a768-00aa006e0529;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RP;4c164200-20c0-11d0-a768-
00aa006e0529;bf967aba-0de6-11d0-a285-
00aa003049e2;RU) (OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-
00aa003049e2;RU) (OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-
00aa003049e2;RU) (OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;bf967aba-0de6-11d0-a285-
00aa003049e2;RU) (OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;bf967aba-0de6-11d0-a285-
00aa003049e2;RU) (OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a86-0de6-11d0-a285-
00aa003049e2;ED) (OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a9c-0de6-11d0-a285-
00aa003049e2;ED) (OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967aba-0de6-11d0-a285-
00aa003049e2;ED) (OA;CIIOID;RCLCRPLO;;4828cc14-1437-45bc-
9b07-ad6f015e5f28;RU) (OA;CIIOID;RCLCRPLO;;
bf967a9c-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RCLCRPLO;;bf967aba-0de6-11d0-a285-
00aa003049e2;RU)
(OA;CIID;RPWPCR;91e647de-d96f-4b70-9557-d63ff4f3ccd8;;PS)
(A;CIID;SDRCWDWOCCDCLCSWRPWPDTLOCR;;;EA)
(A;CIID;LC;;;RU) (A;CIID;SDRCWDWOCCCLCSWRPWPLOCR;;;BA)
```

## Related commands

The following commands enable you to work with security descriptor strings:

- `explain_sd` converts security descriptor in SDDL format to a human-readable form.



- `remove_sd_ace` removes an access control entry (ACE) from a security descriptor.
- `set_sd_owner` sets the owner of a security descriptor.

## bind

Use the `bind` command to bind ADEdit to a domain. Multiple `bind` commands can bind ADEdit to multiple domains in multiple forests. ADEdit must be bound to at least one domain before its commands have any effect on Active Directory or Centrify objects. When ADEdit is bound to multiple domains, its commands can work on any of those domains.

You can use `bind` to bind to any domain for which the DNS can resolve a name and for which you have log-in permission. ADEdit's host computer does not need to be joined to a domain for ADEdit to bind to and work on that domain.

You can optionally specify a server in the domain to bind to, in which case ADEdit binds to that domain controller. If you don't specify a server, ADEdit automatically binds to the closest, fastest domain controller. You can use options to request automatic binding to a global catalog (GC) domain controller or to a writable domain controller.

You can authorize the `bind` connection to a domain controller in the following ways:

- If you provide no `user` or `password` arguments, `bind` uses the user name and password stored in the current Kerberos credential cache on the ADEdit host computer.
- If you provide a `user` argument without the `password` argument, `bind` in interactive mode prompts you for a password, then uses the `user` argument along with your entered password for authorization.
- If you provide a `user` argument and `password` argument, `bind` uses the `user` and `password` arguments for authorization.
- If you specify the `-machine` option, ADEdit authenticates using the credentials for the ADEdit host computer. You cannot provide `user` or `password` arguments if you specify the `-machine` option. Note that you must have read permission on the host's credential files to use this option, so you must typically have root permissions to use the option.

.....

## Zone type

Not applicable

## Syntax

```
bind [-gc] [-write] [-machine] [server@domain] [user
[password]]
```

## Abbreviation

None

## Options

This command takes the following options:

Option	Description
-gc	Requests an automatic binding to a global catalog (GC) domain controller. This option has no effect if there's a domain controller specified using the <i>server</i> argument.
-write	Requests an automatic binding to a writable domain controller. This option has no effect if there's a domain controller specified using the <i>server</i> argument.
-	Binds using the credentials for the ADEdit host computer.
machine	Note that most computer accounts have only read permission, not write permission for Active Directory. To use this option, you must have read permission on the local computer's <i>keytab</i> file and credentials cache. In most cases, only the <i>root</i> user has this right.

## Arguments

This command takes the following arguments:



Argument	Type	Description
		Required. Specifies the domain to bind to.
[ <i>server</i> ]@domain	string	If you want to specify a domain controller to connect to, precede the domain with the name of the domain controller's server followed by the "@" symbol. If you don't specify a domain controller, <code>bind</code> performs an automatic binding to the domain controller that ADEdit determines is most optimal for binding.
[ <i>user</i> ]	string	Optional. Specifies the user name for logging on to the domain controller. If you don't specify this argument and the <code>-machine</code> option is also not present, ADEdit attempts to log on using your current account credentials. If you specify the <code>-machine</code> option, you cannot use this argument.
[ <i>password</i> ]	string	Optional. Requires the <i>user</i> argument. Specifies the password to use when logging on to the domain controller as <i>user</i> .

## Return value

This command returns no value.

## Examples

The following example binds ADEdit to the domain `acme.com`, logging in as `administrator` with the password `#3gEgh^&4`:

```
bind acme.com administrator #3gEgh^&4
```

Note that a password that includes Tcl-special characters such as `$` might trigger character substitution that modifies the password. To ensure that a password isn't altered by the Tcl interpreter, enclose the password in braces (`{}`). For example:

```
bind acme.com maya,garcia {$m113s88}
```

## Related commands

The following commands perform actions related to the `bind` command:



- `get_bind_info` returns information about a domain to which ADEdit is bound.
- `pop` restores the context from the top of ADEdit's context stack to ADEdit.
- `push` saves ADEdit's current context to ADEdit's context stack.
- `show` returns the current context of ADEdit: its bound domains and its currently selected objects.

## clear\_rs\_env\_from\_role

Use the `clear_rs_env_from_role` command to remove the restricted shell environment from the currently selected role that is stored in memory.

The `clear_rs_env_from_role` command does not modify the information stored in Active Directory for the role. If you run this command using ADEdit without saving the role to Active Directory, the change will have no effect on the restricted shell environment stored in Active Directory.

You can only use the `clear_rs_env_from_role` command if the currently selected zone is a classic4 zone. The command does not work in other types of zones.

## Zone type

Classic only

## Syntax

```
clear_rs_env_from_role
```

## Abbreviation

```
crse
```

.....

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
clear_rs_env_from_role
```

This example removes the restricted shell environment from the current role.

## Related commands

The following commands perform actions related to this command:

- [get\\_rs\\_envs](#) returns a Tcl list of restricted shell environments.
- [list\\_rs\\_envs](#) lists to `stdout` the restricted shell environments.
- [new\\_rs\\_env](#) creates a new restricted shell environment and stores it in memory.
- [select\\_rs\\_env](#) retrieves a restricted shell environment from Active Directory and stores it in memory.
- [set\\_rs\\_env\\_for\\_role](#) assigns a restricted shell environment to the current role.

After you have a restricted shell environment stored in memory, you can use the following commands to work with that: restricted shell environment:

• • • • •

- `delete_rs_env` deletes the current restricted shell environment from Active Directory and from memory.
- `get_rse_field` reads a field value from the current restricted shell environment.
- `save_rs_env` saves the restricted shell environment to Active Directory.

## create\_computer\_role

Use the `create_computer_role` command to create a new computer role in Active Directory. The command does *not* store the new computer role in memory nor set it as the currently selected ADEdit computer role. To manage the computer role, you must select it using `select_zone` and then use zone commands to work with the computer role's fields.

ADEdit requires a valid license before the computer role is created. The `create_computer_role` command does an implicit search. The first place it looks is the ADEdit context for a valid license indicator (see the `validate_license` command) for the forest. If an indicator is not in the context, the command checks for a valid license as follows:

- Bind to the global catalog (GC) domain controller, search the forest for the license container and validate the license.
- Bind to the current domain, search for the license container and validate the license.

If it finds a valid license, it stores an indicator in the current context and creates the new computer role. If it does not find a valid license, `create_computer_role` reports "No valid license found" and exits. If the command fails, use `validate_license` to validate the license container explicitly.

To associate role assignments with the new computer role, you must select the computer role, then use `new_role_assignment`.

## Zone type

Hierarchical only

.....

## Syntax

```
create_computer_role computer_role_path group_upn
```

## Abbreviation

ccr

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
computer_role_path	string	Required. Specifies a path to the new computer role. The path consists of the hosting zone's distinguished name followed by a slash and the name of the new computer role.
group_upn	string	Required. Specifies the user principal name (UPN) of a computer group in Active Directory to associate with this computer role. This computer group defines the set of computers in which this computer role functions. The computer group must be available within the computer role's host domain.

## Return value

This command returns no value if it runs successfully.

## Examples

The following example creates a new computer role named `LinuxComputers` in the `global` zone of `acme.com`:

.....

```
create_computer_role  
{CN=global,CN=Zones,CN=Centrify,DC=acme,DC=com/LinuxComputers} linux_computers@acme.com
```

The scope of the computer role is defined by the group named `linux_computers` which is an Active Directory group defined in `acme.com`. To work with the new computer role, you must select it as a zone:

```
select_zone  
"CN=global,CN=Zones,CN=Centrify,DC=acme,DC=com/LinuxComputers"
```

## Related commands

The following command retrieves the computer role from Active Directory and stores it in memory so you can use other commands to work with it.

- `select_zone` retrieves the computer role and stores it in memory.

After you have a computer role selected as a zone, you can use the following commands to view and manage the computer role:

- `new_role_assignment` creates a new role assignment for the selected computer role.
- `list_role_assignments` lists user role assignments for the selected computer role.
- `get_role_assignments` returns a Tcl list of user role assignments for the selected computer role.
- `get_zone_field` retrieves what computer group is associated with the computer role.
- `set_zone_field` sets what computer group is associated with the computer role.
- `delete_zone` deletes the selected computer role from Active Directory and memory.



## create\_zone

Use the `create_zone` command to create a new zone in Active Directory. The command does *not* store the new zone in memory nor set it as the currently selected ADEdit zone. To manage the zone, you must select it using `select_zone` and then use zone commands.

This command can create different types of zones and the zones can use different types of schemas, depending on the schema you are using for Active Directory. Before the zone is created, however, ADEdit checks for a valid license.

The `create_zone` command first checks the ADEdit context for a valid license indicator for the forest. If an indicator is not found in the context, the command checks for a valid license as follows:

- Bind to the global catalog (GC) domain controller, search the forest for the license container and validate the license.
- Bind to the current domain, search for the license container and validate the license.

If the command finds a valid license, it stores an indicator in the current context and creates the new zone. If it does not find a valid license, `create_zone` reports “No valid license found” and exits. If the command fails, use the `validate_license` command to validate the license container explicitly.

**Note** When this command creates a zone, the zone contains predefined roles such as “sftp” and “UNIX Login.” The zone does not, however, contain the role “Windows Login” because ADEdit does not support Windows rights.

## Zone type

Classic and hierarchical

## Syntax

```
create_zone [-ou] [-nonisserversgroup] [-notdelegateanyright] zone_type path [schema_type]
```



## Abbreviation

CZ

## Options

This command takes the following options:

Option	Description
-nonisserversgroup	Creates the new zone without the zone_nis_servers group.
-notdelegateanyright	Creates the new zone but does not set the zone permissions. If you use this option, be sure to set the zone permissions later.
-ou	Creates the new zone as an organizational unit object. If not present, the new zone is created as a container object.  Note that the parent container determines what type of object the zone can be. If the parent container is a generic container object, the zone must be a container object. If the parent container is an organizational unit object, the zone can be either an organizational unit object or a container object.

## Arguments

This command takes the following arguments:

Argument	Type	Description
		Required. The possible values are: <ul style="list-style-type: none"> <li>▪ <b>tree</b> specifies a hierarchical zone that can be a parent or child zone.</li> <li>▪ <b>classic3</b> specifies a classic zone that is compatible with agent version 3 and later.</li> <li>▪ <b>classic4</b> specifies a classic zone that is compatible with agent version 4 and later.</li> </ul>
zone_type	string	<ul style="list-style-type: none"> <li>▪ <b>computer</b> specifies a computer-level zone that consists of a single computer in a hierarchical zone. This zone type is used to support computer-level overrides for user and group profiles and role assignments. It is not applicable in classic zones.</li> <li>▪ <b>classic-computer</b> specifies a computer-level zone that consists of a single computer in a classic zone. This zone type is used to enable you to assign a role to a specific computer in classic zones. It is not applicable in hierarchical zones.</li> </ul>
path	string	Required. Specifies a path to the new zone. The path consists of the new zone's distinguished name (DN) and (if a computer override) the name of the computer.
schema_type	string	<p>Optional. Specifies the type of schema to use for the new zone. The possible values are:</p> <ul style="list-style-type: none"> <li>▪ <b>sfu</b> specifies the Microsoft Services For UNIX schema. This setting can be used for tree, classic3, and classic4 zone types. If it's used for a hierarchical zone, it can only be the root of the zone hierarchy.</li> <li>▪ <b>std</b> specifies the dynamic schema. This setting can be used for all zone types. This is the default schema unless ADEdit detects the RFC2307 schema.</li> <li>▪ <b>rfc</b> specifies the RFC2307 schema. This setting can be used for all zone types. This is the default schema if ADEdit detects that RFC2307 is installed and the domain is at Windows Server 2003 functional level.</li> </ul> <p>If none of these values is present, the default is either <b>std</b> or <b>rfc</b> as described above.</p>

.....

## Return value

This command returns no value if it runs successfully.

## Examples

The following examples illustrate how to create a classic zone, hierarchical zone, and computer-specific zone in Centrify Server Suite 2012 and later.

### Classic zone

The following command creates a classic zone named `finance` in the `centrify` organizational unit in the `acme.com` domain that uses the dynamic schema (`std`):

The following command creates a classic zone named `finance` in the `centrify` organizational unit in the `acme.com` domain that uses the dynamic schema (`std`):

```
create_zone classic4
"CN=finance,OU=Centrify,DC=acme,DC=com" std
```

### Hierarchical zone

The following command creates a new hierarchical parent zone named `finance` in the `zones` container in the `centrify` organizational unit in the `acme.com` domain:

```
create_zone tree
"CN=finance,CN=Zones,OU=Centrify,DC=acme,DC=com" std
```

To make the `finance` zone a child zone within a `global` zone already created in the same container, OU, and domain, you would next select `finance` to make it the currently selected zone, then use `set_zone_field` (`szf`) to specify the `global` zone as its parent, and the save `finance`. For example:

```
select_zone "CN=finance,CN=Zones,OU=UNIX,DC=acme,DC=com"
szf parent "CN=global,CN=Zones,OU=UNIX,DC=acme,DC=com"
save_zone
```

### Computer-specific zone

The following command creates a computer-specific zone for the computer `srv1` in the `apache` zone, which is a child of the `global` zone, in the `zones`

• • • • •

container in the `centrifly` organizational unit in the `acme.com` domain.

```
create_zone computer  
svrl.acme.com@CN=apache,CN=global,CN=Zones,OU=Centrifly,DC=acme,DC=com
```

## Related commands

Before you use this command, you must bind to one or more Active Directory domains. The following command enables you to store a newly created zone in memory:

- `select_zone` retrieves a zone from Active Directory and stores it in memory.

After you have created a new zone and stored it in memory, you can use the following commands to work with that zone:

- `delegate_zone_right` delegates a zone use right to a specified user or computer.
- `delete_zone` deletes the selected zone from Active Directory and memory.
- `get_child_zones` returns a Tcl list of child zones, computer roles, or computer zones.
- `get_zone_field` reads a field value from the currently selected zone.
- `get_zone_nss_vars` returns the NSS substitution variable for the selected zone.
- `save_zone` saves the selected zone with its current settings to Active Directory.
- `set_zone_field` sets a field value in the currently selected zone.

## delegate\_zone\_right

Use the `delegate_zone_right` command to delegate an administrative right for the currently selected zone to a security principal (user or group). Zone rights allow a user or group to use and manage zone properties, including computer-specific zone properties and computer roles.

.....

## Zone type

Classic and hierarchical

## Syntax

```
delegate_zone_right right principal_upn
```

## Abbreviation

None.

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
		Required. Specifies the right to delegate. Possible values:
		<ul style="list-style-type: none"><li>▪ <b>add_computer_role</b>: The right to add computer roles to the zone.</li><li>▪ <b>add_computer_zone</b>: The right to add computer-specific zones.</li><li>▪ <b>add_group</b>: The right to add groups to the zone.</li><li>▪ <b>add_nismap</b>: The right to add NIS maps to the zone.</li><li>▪ <b>add_remove_nismap_entry</b>: The right to add or remove NIS map entries.</li><li>▪ <b>add_user</b>: The right to add users to the zone.</li></ul>
right	string	



Argument	Type	Description
		<ul style="list-style-type: none"><li>▪ <b>add_user_group_to_computer_zone:</b> The right to add user and group overrides to the selected computer-specific zone.</li><li>▪ <b>change_user:</b> The right to modify user profiles in the zone.</li><li>▪ <b>change_group:</b> The right to modify group profiles in the zone.</li></ul>
		<ul style="list-style-type: none"><li>▪ <b>change_computer:</b> The right to modify computer profiles in the zone.</li><li>▪ <b>change_zone:</b> The right to change zone properties.</li><li>▪ <b>delegate_permission_for_computer_zone:</b> The right to delegate permissions to other users for computer-specific zones.</li></ul>
		<ul style="list-style-type: none"><li>▪ <b>delete_computer:</b> The right to remove computers from the zone.</li><li>▪ <b>delete_computer_role:</b> The right to delete computer roles in the zone.</li><li>▪ <b>delete_computer_zone:</b> The right to delete computer-specific zones.</li></ul>
right (continued)	string (continued)	<ul style="list-style-type: none"><li>▪ <b>delete_group:</b> The right to remove groups from the zone.</li><li>▪ <b>delete_user:</b> The right to remove users from the zone.</li><li>▪ <b>delete_user_group_from_computer_zone:</b> The right to delete user and group overrides from the selected computer-specific zone.</li></ul>
		<ul style="list-style-type: none"><li>▪ <b>delete_zone:</b> The right to remove the zone.</li><li>▪ <b>enable_dz:</b> The right to initialize authorization (privilege elevation service) data. This right is only applicable in classic zones.</li><li>▪ <b>import:</b> The right to import users and groups</li></ul>



Argument	Type	Description
		into the zone.
		<ul style="list-style-type: none"><li>▪ <b>join</b>: The right to join computers to the zone.</li><li>▪ <b>manage_role_assignments</b>: The right to modify the roles assigned in zones, computer-specific zones, and computer roles.</li><li>▪ <b>manage_roles_and_rights</b>: The right to modify role definitions and access rights.</li></ul>
		<ul style="list-style-type: none"><li>▪ <b>modify_computer_role</b>: The right to modify computer role entries. This right is not applicable in classic zones.</li><li>▪ <b>modify_nismap_entry</b>: The right to modify NIS map entries.</li><li>▪ <b>modify_user_group_in_computer_zone</b>: The right to modify user and group overrides in the selected computer-specific zone.</li></ul>
right (continued)	string (continued)	<ul style="list-style-type: none"><li>▪ <b>nisservers</b>: The right to allow computers to respond to NIS client requests.</li><li>▪ <b>remove_nismap</b>: The right to remove NIS maps.</li></ul>
principal_ upn	string	Required. Specifies the user principal name (UPN) of a user or group in Active Directory to delegate the specified right to.

## Return value

This command returns no value if it runs successfully.

## Examples

```
delegate_zone_right add_user adam.avery@acme.com
```

This example delegates the right to add users to the currently selected zone to the Active Directory user Adam Avery.



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone to work with:

- `create_zone` creates a new zone in Active Directory.
- `get_zones` returns a Tcl list of all zones within a specified domain.
- `select_zone` retrieves a zone from Active Directory and stores it in memory.

After you have a zone stored in memory, you can use the following commands to work with that zone:

- `delegate_zone_right` delegates a zone use right to a specified user or computer.
- `delete_zone` deletes the selected zone from Active Directory and memory.
- `get_child_zones` returns a Tcl list of child zones, computer roles, or computer zones.
- `get_zone_field` reads a field value from the currently selected zone.
- `get_zone_nss_vars` returns the NSS substitution variable for the selected zone.
- `save_zone` saves the selected zone with its current settings to Active Directory.
- `set_zone_field` sets a field value in the currently selected zone.

## delete\_dz\_command

Use the `delete_dz_command` command to delete the currently selected privileged command from Active Directory and from memory. You cannot use other commands to manage privileged commands after deletion because there will be no currently selected command in memory.

.....

## Zone type

Classic and hierarchical

## Syntax

```
delete_dz_command
```

## Abbreviation

```
dldzc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_dz_command
```

This example deletes the currently selected command from Active Directory and from memory.



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a UNIX command to work with:

- `get_dz_commands` returns a Tcl list of UNIX commands in the current zone.
- `list_dz_commands` lists to `stdout` the UNIX commands in the current zone.
- `new_dz_command` creates a new UNIX command and stores it in memory.
- `select_dz_command` retrieves a UNIX command from Active Directory and stores it in memory.

After you have a UNIX command stored in memory, you can use the following commands to work with that command:

- `get_dzc_field` reads a field value from the currently selected command.
- `save_dz_command` saves the selected command with its current settings to Active Directory.
- `set_dzc_field` sets a field value in the currently selected command.

## `delete_local_group_profile`

Use the `delete_local_group_profile` command to delete a local UNIX or Linux group that has a profile defined in the current zone. When you delete a group, the group's zone object is deleted, but the group's entry in the local `/etc/group` file is retained.

## Zone type

Hierarchical only.

.....

## Syntax

```
delete_local_group_profile group_name
```

## Abbreviation

dl1gp

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
group_name	string	Required. Specifies the UNIX name of the local group to delete from the zone.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_local_group_profile marketing
```

This example deletes the zone object for the local group `marketing`. The entry for `marketing` in the local `/etc/group` file is retained.



## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.



- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## delete\_local\_user\_profile

Use the `delete_local_user_profile` command to delete a local UNIX or Linux user that has a profile defined in the current zone. When you delete a user, the user's zone object is deleted, but the user's entry in the local `/etc/passwd` file is retained.

### Zone type

Hierarchical only.

### Syntax

```
delete_local_user_profile user_name
```

### Abbreviation

`dllup`

### Options

This command takes no options.

.....

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>user_name</code>	string	Required. Specifies the UNIX name of the local user to delete from the zone.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_local_user_profile anton.splieth
```

This example deletes the zone object for the local user `anton.splieth`. The entry for `anton.splieth` in the local `/etc/passwd` file is retained.

## Related commands

The following related ADEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- `delete_local_group_profile` deletes a local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.



- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## `delete_map_entry`

Use the `delete_map_entry` command to delete an entry from the currently selected NIS map stored in memory. The `delete_map_entry` command changes the NIS map in memory and in Active Directory. You do not need to save the NIS map for the deleted entry to take effect in Active Directory.

.....

## Zone type

Not applicable

## Syntax

```
delete_map_entry key:index
```

## Abbreviation

dlme

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
key:index	string	Required. Specifies the key of the NIS map entry to delete followed by a colon (:) and the index number of the key.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_map_entry calla:1
```



This example deletes the NIS map entry with the key value “calla” and index number 1 from the currently selected NIS map.

## Related commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and select the NIS map to work with:

- `get_nis_maps` returns a Tcl list of NIS maps in the currently selected zone.
- `list_nis_maps` lists to `stdout` all of the NIS maps in the currently selected zone.
- `new_nis_map` creates a new NIS map and stores it in memory.
- `select_nis_map` retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map’s entries:

- `get_nis_map` or `get_nis_map_with_comment` returns a Tcl list of the map entries in the currently selected NIS map.
- `get_nis_map_field` reads a field value from the currently selected NIS map.
- `list_nis_map` or `list_nis_map_with_comment` lists to `stdout` the map entries in the currently selected NIS map.
- `add_map_entry` or `add_map_entry_with_comment` adds an map entry to the currently selected NIS map.

## delete\_nis\_map

Use the `delete_nis_map` command to delete the currently selected NIS map from Active Directory and from memory. You cannot use other commands to manage the NIS map after deletion because there will be no currently selected map in memory.

.....

## Zone type

Not applicable

## Syntax

```
delete_nis_map
```

## Abbreviation

```
dlnm
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_nis_map
```

This example deletes the currently selected NIS map from Active Directory and from memory.



## Related commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and select the NIS map to work with:

- `get_nis_maps` returns a Tcl list of NIS maps in the currently selected zone.
- `list_nis_maps` lists to stdout of all NIS maps in the currently selected zone.
- `new_nis_map` creates a new NIS map and stores it in memory.
- `select_nis_map` retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map's entries:

- `add_map_entry` or `add_map_entry_with_comment` adds an entry to the currently selected NIS map.
- `delete_map_entry` removes an entry from the currently selected NIS map.
- `get_nis_map` or `get_nis_map_with_comment` returns a Tcl list of the entries in the currently selected NIS map.
- `get_nis_map_field` reads a field value from the currently selected NIS map.
- `list_nis_map` or `list_nis_map_with_comment` lists to stdout of the entries in the currently selected NIS map.

## delete\_object

Use the `delete_object` command to delete the currently selected Active Directory object from Active Directory and from memory. You cannot use other commands to manage the object after deletion because there will be no currently selected Active Directory object in memory.

**Note** Do NOT use the `delete_object` command to delete an Active Directory user or group that has been provisioned. If you use `delete_`



**Note** object to delete a provisioned user or group, you create orphan user or group UNIX data objects. Instead, use the `delete_zone_user` or `delete_zone_group` command. In addition, you would use the `select_zone_user` and `select_zone_group` rather than `select_object` to select the user or group. For information about displaying orphan accounts, see the `list_zone_users` and `list_zone_groups`.

## Zone type

Not applicable

## Syntax

```
delete_object
```

## Abbreviation

dlo

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.



## Examples

`delete_object`

This example deletes the currently selected Active Directory object from Active Directory and from memory.

## Related commands

Before you use this command, you must have a currently selected Active Directory object stored in memory. The following commands enable you to view and select the object to work with:

- `get_objects` performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of matching objects.
- `new_object` creates a new Active Directory object and stores it in memory.
- `select_object` retrieves an object with its attributes from Active Directory and stores it in memory.

After you have an Active Directory object stored in memory, you can use other commands to work with that object's attributes, or the following commands to delete or save information for the object:

- `delete_sub_tree` deletes an Active Directory object and all of its children from Active Directory.
- `save_object` saves the selected Active Directory object with its current settings to Active Directory.

## `delete_pam_app`

Use the `delete_pam_app` command to delete the currently selected PAM application from Active Directory and from memory. You cannot use other commands to manage the PAM application after deletion because there will be no currently selected PAM application in memory.

.....

## Zone type

Classic and hierarchical

## Syntax

```
delete_pam_app
```

## Abbreviation

```
dlpam
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_pam_app
```

This example deletes the currently selected PAM application from Active Directory and from memory.



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. After you have a zone stored in memory, you can use the following commands to view and select the PAM application to work with:

- `get_pam_apps` returns a Tcl list of PAM application rights in the current zone.
- `list_pam_apps` lists to `stdout` all PAM application rights in the current zone.
- `new_pam_app` creates a new PAM application right and stores it in memory.
- `select_pam_app` retrieves a PAM application right from Active Directory and stores it in memory

After you have a PAM application stored in memory, you can use the following commands to work with that PAM application's attributes, delete the PAM application, or save information for the PAM application:

- `delete_pam_app` deletes the selected PAM application right from Active Directory and from memory.
- `get_pam_field` reads a field value from the currently selected PAM application right.
- `save_pam_app` saves the selected PAM application right with its current settings to Active Directory.
- `set_pam_field` sets a field value in the currently selected PAM application right.

## delete\_role

Use the `delete_role` command to delete the currently selected role from Active Directory and from memory. You cannot use other commands to manage the role after deletion because there will be no currently selected role in memory.

.....

## Zone type

Classic and hierarchical

## Syntax

```
delete_role
```

## Abbreviation

```
d1r
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_role
```

This example deletes the currently selected role from Active Directory and from memory.



## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with:

- `get_roles` returns a Tcl list of roles in the current zone.
- `list_roles` lists to `stdout` all roles in the currently selected zone.
- `new_role` creates a new role and stores it in memory.
- `select_role` retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with that role:

- `add_command_to_role` adds a UNIX command to the currently selected role.
- `add_pamapp_to_role` adds a PAM application to the currently selected role.
- `get_role_apps` returns a Tcl list of the PAM applications associated with the role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the role.
- `get_role_field` reads a field value from the currently selected role.
- `list_role_rights` lists to `stdout` all UNIX commands and PAM applications associated with the role.
- `remove_command_from_role` removes a UNIX command from the currently selected role.
- `remove_pamapp_from_role` removes a PAM application from the currently selected role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the currently selected role.

.....

## delete\_role\_assignment

Use the `delete_role_assignment` command to delete the currently selected role assignment from Active Directory and from memory. You cannot use other commands to manage the role assignment after deletion because there will be no currently selected role assignment in memory.

### Zone type

Classic and hierarchical

### Syntax

```
delete_role_assignment
```

### Abbreviation

`dlra`

### Options

This command takes no options.

### Arguments

This command takes no arguments.

### Return value

This command returns nothing if it runs successfully.



## Examples

```
delete_role_assignment
```

This example deletes the currently selected role assignment from Active Directory and from memory.

## Related commands

Before you use this command, you must have a currently selected role assignment stored in memory. The following commands enable you to view and select the role assignment to work with:

- `get_role_assignments` returns a Tcl list of role assignments in the current zone.
- `list_role_assignments` lists to `stdout` all role assignments in the currently selected zone.
- `new_role_assignment` creates a new role assignment and stores it in memory.
- `select_role_assignment` retrieves a role assignment from Active Directory and stores it in memory.

After you have a role assignment stored in memory, you can use other commands to work with that role assignment's fields, or the following commands to save information for the role assignment:

- `save_role_assignment` saves the selected role assignment with its current settings to Active Directory.

## delete\_rs\_command

Use the `delete_rs_command` command to delete the currently selected restricted shell command from Active Directory and from memory. After you run this command, you cannot run subsequent ADEdit commands for restricted shell commands because there will be no currently selected restricted shell command available in memory.

.....

## Zone type

Classic only

## Syntax

```
delete_rs_command
```

## Abbreviation

```
dlrsc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_rs_command
```

This example deletes the currently selected restricted shell command from Active Directory and from memory.



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the restricted shell command to work with:

- `get_rs_commands` returns a Tcl list of restricted shell commands in the current zone.
- `list_rs_commands` lists to `stdout` the restricted shell commands in the current zone.
- `new_rs_command` creates a new restricted shell command and stores it in memory.
- `select_rs_command` retrieves a restricted shell command from Active Directory and stores it in memory.

After you have a restricted shell command stored in memory, you can use the following commands to work with that restricted shell:

- `get_rsc_field` reads a field value from the currently selected command.
- `save_rs_command` saves the selected command with its current settings to Active Directory.
- `set_rsc_field` sets a field value in the currently selected command.

## delete\_rs\_env

Use the `delete_rs_env` command to delete the currently selected restricted environment from Active Directory and from memory. After you run this command, you cannot run subsequent ADEdit commands for a restricted shell environment because there will be no currently selected restricted shell environment available in memory.

## Zone type

Classic only

.....

## Syntax

```
delete_rs_env
```

## Abbreviation

```
dlrse
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_rs_env
```

This example deletes the currently selected RSE from Active Directory and from memory.

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with restricted shell environments:



- `get_rs_envs` returns a Tcl list of restricted shell environments.
- `list_rs_envs` lists to `stdout` the restricted shell environments.
- `new_rs_env` creates a new restricted shell environment and stores it in memory.
- `select_rs_env` retrieves a restricted shell environment from Active Directory and stores it in memory.

After you have a restricted shell environment stored in memory, you can use the following commands to work with its fields:

- `get_rse_field` reads a field value from the current restricted shell environment.
- `save_rs_env` saves the restricted shell environment to Active Directory.
- `set_rse_field` sets a field value in the current restricted shell environment.

## delete\_sub\_tree

Use the `delete_sub_tree` command to delete an object and all of its child objects from Active Directory. Only child objects that are in the same container as the specified parent object are deleted. Child objects in other containers are not deleted.

**WARNING:** This is a very powerful command, and can cause a lot of damage if used incorrectly. It's similar to running `rm -rf *` in UNIX.

In interactive mode, ADEdit prompts you for confirmation before executing this command. If you use this command in a script, ADEdit does not prompt for confirmation. You should use caution before using this command in a script.

This command can be used on any Active Directory object, including a container, OU, computer object, group or user. However, it is especially useful for deleting a corrupted zone. You'd normally use `select_zone` and then `delete_zone` to delete a zone. If the zone is damaged, though, `select_zone` might not work. In that case, `delete_sub_tree` will do the job.



If the zone is a hierarchical zone, this command deletes only the child zones in the same container as the parent zone. If there are any child zones in other containers, they are not deleted.

## Zone type

Classic and hierarchical

## Syntax

```
delete_sub_tree dn
```

## Abbreviation

None.

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
dn	DN	Required. Specifies the distinguished name of the object (with all of its children) to remove from Active Directory.

## Return value

This command returns nothing if it runs successfully.



## Examples

```
delete_sub_tree  
"CN=marketing,CN=Zones,CN=Centrify,DC=acme,DC=com"
```

This example deletes the currently selected "marketing" zone with all of its children from Active Directory.

## Related commands

The following commands enable you to view and manage the Active Directory object to work with:

- [delete\\_object](#) deletes the selected Active Directory object from Active Directory and from memory.
- [get\\_objects](#) performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of matching objects.
- [new\\_object](#) creates a new Active Directory object and stores it in memory.
- [save\\_object](#) saves the selected Active Directory object with its current settings to Active Directory.
- [select\\_object](#) retrieves an object with its attributes from Active Directory and stores it in memory.

The following commands enable you to view and manage Active Directory object attributes:

- [add\\_object\\_value](#) adds a value to a multi-valued field attribute of the currently selected Active Directory object.
- [get\\_object\\_field](#) reads a field value from the currently selected Active Directory object.
- [remove\\_object\\_value](#) removes a value from a multi-valued field attribute of the currently selected Active Directory object.
- [set\\_object\\_field](#) sets a field (attribute) value in the currently selected Active Directory object.



## delete\_zone

Use the `delete_zone` command to delete the currently selected zone from Active Directory and from memory. After you run this command, you cannot run subsequent ADEdit commands for zones because there will be no currently selected zone available in memory.

This command performs an LDAP sub-tree deletion operation. Only child zones that are in the same container as the specified parent zone are deleted. Child zones that are located in other containers are not deleted. Child zones that are based on pointers defined in the child zone are not deleted. For more information about deleting sub-tree objects, see [delete\\_sub\\_tree](#).

In interactive mode, ADEdit prompts you for confirmation before executing this command. If you use this command in a script, ADEdit does not prompt for confirmation. You should use caution before using this command in a script.

## Zone type

Classic and hierarchical

## Syntax

```
delete_zone
```

## Abbreviation

```
dlz
```

## Options

This command takes no options.

.....

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_zone
```

This example deletes the currently selected zone or computer role from Active Directory and from memory.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the zone to work with:

- [create\\_zone](#) creates a new zone in Active Directory.
- [get\\_zones](#) returns a Tcl list of all zones within a specified domain.
- [select\\_zone](#) retrieves a zone from Active Directory and stores it in memory as the currently selected zone.

After you have a zone stored in memory, you can use the following commands to work with that zone:

- [delegate\\_zone\\_right](#) delegates an administrative right to a specified user or group.
- [get\\_child\\_zones](#) returns a Tcl list of child zones, computer roles, or computer zones for the current zone.
- [get\\_zone\\_field](#) reads a field value from the currently selected zone.
- [set\\_zone\\_field](#) sets a field value in the currently selected zone.



- `get_zone_nss_vars` returns the NSS substitution variable for the selected zone.
- `save_zone` saves the selected zone with its current settings to Active Directory.

## delete\_zone\_computer

Use the `delete_zone_computer` command to delete the currently selected zone computer profile from Active Directory and from memory. After you run this command, you cannot run subsequent ADEdit commands for zone computer profiles because there will be no currently selected zone computer profile available in memory. This command only deletes the zone profile for the computer. It does not delete the Active Directory computer account.

## Zone type

Classic and hierarchical

## Syntax

```
delete_zone_computer [-all]
```

## Abbreviation

dlzc

## Options

This command takes the following option:

Option	Description
-all	Removes the corresponding computer-specific zone profile if the selected computer is a computer-specific override zone.

.....

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_zone_computer
```

This example deletes the currently selected zone computer profile from Active Directory and from memory.

## Related commands

Before you use this command, you must have a currently selected zone computer profile stored in memory. The following commands enable you to view and select the zone computer profile to work with:

- [get\\_zone\\_computers](#) returns a Tcl list of the Active Directory names of all zone computer profiles in the current zone.
- [list\\_zone\\_computers](#) lists to stdout all zone computer profiles in the current zone.
- [new\\_zone\\_computer](#) creates a new zone computer profile and stores it in memory.
- [select\\_zone\\_computer](#) retrieves a zone computer profile from Active Directory and stores it in memory.

After you have a zone computer stored in memory, you can use the following commands to work with that zone computer:

- [get\\_zone\\_computer\\_field](#) reads a field value from the currently selected zone computer profile.



- `set_zone_computer_field` sets a field value in the currently selected zone computer profile.
- `save_zone_computer` saves the selected zone computer profile with its current settings to Active Directory.

## delete\_zone\_group

Use the `delete_zone_group` command to delete the currently selected zone group profile from Active Directory and from memory. After you run this command, you cannot run subsequent ADEdit commands for zone groups because there will be no currently selected zone group available in memory.

### Zone type

Classic and hierarchical

### Syntax

```
delete_zone_group
```

### Abbreviation

```
dlzg
```

### Options

This command takes no options.

### Arguments

This command takes no arguments.

• • • • •

## Return value

This command returns nothing if it runs successfully.

## Examples

```
delete_zone_group
```

This example deletes the currently selected zone group from Active Directory and from memory.

## Related commands

Before you use this command, you must have a currently selected zone group stored in memory. The following commands enable you to view and select the zone group to work with:

- [get\\_zone\\_groups](#) returns a Tcl list of the Active Directory names of all zone groups in the current zone.
- [list\\_zone\\_groups](#) lists to stdout all zone groups in the current zone.
- [new\\_zone\\_group](#) creates a new zone group and stores it in memory.
- [select\\_zone\\_group](#) retrieves a zone group from Active Directory and stores it in memory.

After you have a zone group stored in memory, you can use the following commands to work with that zone group:

- [get\\_zone\\_group\\_field](#) reads a field value from the currently selected zone group.
- [save\\_zone\\_group](#) saves the selected zone group with its current settings to Active Directory.
- [set\\_zone\\_group\\_field](#) sets a field value in the currently selected zone group.



## delete\_zone\_user

Use the `delete_zone_user` command to delete the currently selected zone user profile from Active Directory and from memory. After you run this command, you cannot run subsequent ADEdit commands for zone users because there will be no currently selected zone user available in memory.

### Zone type

Classic and hierarchical

### Syntax

```
delete_zone_user
```

### Abbreviation

dlzu

### Options

This command takes no options.

### Arguments

This command takes no arguments.

### Return value

This command returns nothing if it runs successfully.



## Examples

```
delete_zone_user
```

deletes the currently selected zone user from Active Directory and from memory.

## Related commands

Before you use this command, you must have a currently selected zone user stored in memory. The following commands enable you to view and select the zone user to work with:

- `get_zone_users` returns a Tcl list of the Active Directory names of all zone users in the current zone.
- `list_zone_users` lists to stdout all zone users in the current zone.
- `new_zone_user` creates a new zone user and stores it in memory.
- `select_zone_user` retrieves a zone user from Active Directory and stores it in memory.

After you have a zone user stored in memory, you can use the following commands to work with that zone user:

- `get_zone_user_field` reads a field value from the currently selected zone user.
- `save_zone_user` saves the selected zone user with its current settings to Active Directory.
- `set_zone_user_field` sets a field value in the currently selected zone user.

## dn\_from\_domain

Use the `dn_from_domain` command to convert a specified domain name in dotted form (acme.com, for example) to a distinguished name (DN). This conversion doesn't require lookup in Active Directory. The command performs a simple text conversion.

.....

## Zone type

Not applicable

## Syntax

```
dn_from_domain domain_name
```

## Abbreviation

dnfd

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
domain_name	string	Required. Specifies a dotted domain name (acme.com, for example)

## Return value

This command returns a domain name as a distinguished name.

## Examples

```
dn_from_domain acme.com
```

This example returns the domain name in this form: dc=acme,dc=com



## Related commands

The following commands convert information from one format to another:

- `domain_from_dn` converts a domain's distinguished name (DN) to a dotted name.
- `dn_to_principal` returns the `SAMAccount@domain` name or user principal name (UPN) for a security principal.

## dn\_to\_principal

Use the `dn_to_principal` command to specify the distinguished name (DN) of a security principal (user, computer, or group). The command searches Active Directory for the principal, and if the principal is found, the command returns the `SAMAccount@domain` name of the principal. Optionally, you can also use this command to return the user principal name (UPN) for the principal.

## Zone type

Not applicable

## Syntax

```
dn_to_principal [-upn] principal_dn
```

## Abbreviation

dntp

## Options

This command takes the following option:



---

Option	Description
-upn	Returns the principal name in user principal name (UPN) format, not the default SAMAccount@domain format.

---

## Arguments

This command takes the following argument:

---

Argument	Type	Description
principal_dn	string	Required. Specifies the distinguished name (DN) of a security principal.

---

## Return value

This command returns the `SAMAccount@domain` name or (optionally) the user principal name (UPN) of a security principal. If the command doesn't find the specified security principal in Active Directory, it presents a message that it didn't find the principal.

## Examples

```
dn_to_principal cn=brenda butler,cn=users,dc=acme,dc=com
```

This example returns: `brenda.butler@acme.com`

## Related commands

The following commands search for security principals in Active Directory:

- `principal_to_dn` searches Active Directory for a user principal name (UPN) and, if found, returns the corresponding distinguished name (DN).
- `principal_from_sid` searches Active Directory for an SID and returns the security principal associated with the SID.

.....

## domain\_from\_dn

Use the `domain_from_dn` command takes a distinguished name (DN) that contains a domain and returns the domain name in dotted form (acme.com, for example). This conversion doesn't require lookup in Active Directory. The command performs a simple text conversion.

### Zone type

Not applicable

### Syntax

```
domain_from_dn dn
```

### Abbreviation

dfdn

### Options

This command takes no options.

### Arguments

This command takes the following argument:

Argument	Type	Description
dn	string	Required. Specifies a distinguished name that contains a domain.

.....

## Return value

This command returns a domain name in dotted form such as `acme.com`. If the distinguished name doesn't contain domain component (DC) values, the command returns a notice that the DC values are missing.

## Examples

```
dfdn cn=johndoe,cn=users,dc=acme,dc=com
```

This example returns: `acme.com`

## Related commands

The following command converts information from one format to another:

- `dn_from_domain` converts a domain's dotted name to a distinguished name.

## explain\_sd

Use the `explain_sd` command to specify a security descriptor (SD) in security descriptor description language (SDDL) form and returns a human-readable form of the security descriptor.

## Zone type

Not applicable

## Syntax

```
explain_sd sddl_string
```

.....

## Abbreviation

None.

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
sddl_string	string	Required. Specifies a security descriptor in SDDL format.

## Return value

This command returns text that describes the supplied security descriptor in human-readable form.

## Examples

```
explain_sd O:DAG:DAD:AI(A;;SDRCWDWOCDCCLCSWRPWPDTLOCR;;;SY)
(A;;RCWDWOCDCCLCSWRPWPLOCR;;;DA)
(OA;;CCDC;bf967aba-0de6-11d0-a285-00aa003049e2;;;AO)
(OA;;CCDC;bf967a9c-0de6-11d0-a285-00aa003049e2;;;AO)
(OA;;CCDC;bf967aa8-0de6-11d0-a285-00aa003049e2;;;PO)
(A;;RCLCRPLO;;;AU) (OA;;CCDC;4828cc14-1437-45bc-
9b07-ad6f015e5f28;;;AO) (OA;CIIOID;RP;4c164200-20c0-11d0-
a768-00aa006e0529;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RP;4c164200-20c0-11d0-a768-
00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
```

.....

```
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-00a0c983f608;bf967a86-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-00a0c983f608;bf967a9c-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-00a0c983f608;bf967aba-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RCLCRPLO;;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU) (OA;CIIOID;RCLCRPLO;;bf967a9c-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RCLCRPLO;;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIID;RPWPCR;91e647de-d96f-4b70-9557-d63ff4f3ccd8;;PS)
(A;CIID;SDRCWDWOCCLCSWRPWPDTLOCR;;;EA)
(A;CIID;LC;;;RU) (A;CIID;SDRCWDWOCCLCSWRPWPLOCR;;;BA)
```

This example returns the security descriptor information in readable form:

```
Owner: Domain Admins
Group: Domain Admins
Dacl: inherit supported,
  Allow |  | delete,read SD,write DACL,change owner,create
child,delete child,list children,self write,read
property,write property,delete tree,list object,control
access, |  | System
  Allow |  | read SD,write DACL,change owner,create
child,delete child,list children,self write,read
property,write property,list object,control access, |  |
Domain Admins
  Allow |  | create child,delete child, | User |  | Account
operators
  Allow |  | create child,delete child, | Group |  | Account
operators
  Allow |  | create child,delete child, | Print-Queue |  |
Print operators
```

.....

```
Allow | | read SD,list children,read property,list
object, | | | Authenticated users
Allow | | create child,delete child, | inetOrgPerson | |
Account operators
Allow | inherit,inherit ony,inherited, | read property, |
User-Account-Restrictions | inetOrgPerson | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
User-Account-Restrictions | User | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
User-Logon | inetOrgPerson | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
User-Logon | User | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
Membership | inetOrgPerson | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
Membership | User | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
General-Information | inetOrgPerson | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
General-Information | User | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
RAS-Information | inetOrgPerson | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
RAS-Information | User | pre win2k
Allow | inherit,inherit ony,inherited, | read property, |
Token-Groups | Computer | Enterprise Domain Controllers
Allow | inherit,inherit ony,inherited, | read property, |
Token-Groups | Group | Enterprise Domain Controllers
Allow | inherit,inherit ony,inherited, | read property, |
Token-Groups | User | Enterprise Domain Controllers
Allow | inherit,inherit ony,inherited, | read SD,list
children,read property,list object, | | inetOrgPerson |
pre win2k
Allow | inherit,inherit ony,inherited, | read SD,list
children,read property,list object, | | Group | pre win2k
Allow | inherit,inherit ony,inherited, | read SD,list
children,read property,list object, | | User | pre win2k
Allow | inherit,inherited, | read property,write
property,control access, | Private-Information | | Self
Allow | inherit,inherited, | delete,read SD,write
DACL,change owner,create child,delete child,list
children,self write,read property,write property,delete
tree,list object,control access, | | | Enterprise Admins
Allow | inherit,inherited, | list children, | | | pre
win2k
```

.....

Allow | inherit, inherited, | delete, read SD, write  
DACL, change owner, create child, list children, self  
write, read property, write property, list object, control  
access, | | | Administrators

## Related commands

The following commands enable you to work with security descriptor strings:

- `remove_sd_ace` removes an access control entry (ACE) from a security descriptor.
- `add_sd_ace` adds an access control entry to a security descriptor.
- `set_sd_owner` sets the owner of a security descriptor.

## forest\_from\_domain

Use the `forest_from_domain` command to retrieve the forest name based on the domain name. The command also stores the retrieved forest name in memory.

## Zone type

Not applicable

## Syntax

```
forest_from_domain [-nocache] <domain>
```

## Abbreviation

ffd



## Options

This command takes the following option:

Option	Description
<code>nocache</code>	Use this option to force fetch the forest name from Active Directory instead of reading the forest name from in memory.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>domain</code>	string	Required. Specifies the domain for which you want to retrieve the forest.

## Return value

This command returns the forest name (in upper case text).

## Examples

```
>forest_from_domain 5027f1d2.test  
5027F1D1.TEST  
>ffd 5027f1d2.test  
5027F1D1.TEST
```

## get\_adinfo

Use the `get_adinfo` command to return information about the current join state for the ADEdit host computer. The command returns information about the joined domain, the joined zone, or the name the host computer is joined under.

.....

## Zone type

Not applicable

## Syntax

```
get_ainfo domain|zone|host
```

## Abbreviation

ainfo

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
		Required. The possible values are:
		■ <b>domain</b> returns the name of the currently joined domain.
domain zone host	string	■ <b>zone</b> returns the distinguished name of the currently joined zone.
		■ <b>host</b> returns the name under which the ADEdit host computer is joined.

.....

## Return value

This command returns a domain name, zone name, or computer name depending on the provided argument.

## Examples

```
get_ainfo domain
```

This example returns the joined domain. For example: `acme.com`

```
get_ainfo zone
```

This example returns the path to the joined zone. For example:

```
CN=default,CN=Zones,CN=Centrify,CN=Program  
Data,DC=acme,DC=com
```

## Related commands

None.

## get\_bind\_info

Use the `get_bind_info` command to return information about one of ADEdit's currently bound domains. The command can return the name of the domain's forest, the name of the server bound within the domain, the security identifier (SID) of the domain, and the functional level of the domain or the domain's forest.

## Zone type

Not applicable



## Syntax

```
get_bind_info domain forest|server|sid|domain_level|forest_level
```

## Abbreviation

gbi

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
domain	string	Required. Specifies the name of the domain for which to get information.
forest   server   sid   domain_level   forest_level	string	Required. The possible values are: <ul style="list-style-type: none"><li>■ <b>forest</b> returns the name of the forest that contains the bound domain.</li><li>■ <b>server</b> returns the name of the domain server to which ADEdit is bound in the domain.</li><li>■ <b>sid</b> returns the SID (security identifier) of the bound domain.</li><li>■ <b>domain_level</b> returns the functional level of the bound domain, represented by an integer value:<ul style="list-style-type: none"><li>-1: unknown functional level</li><li>0: Windows 2000 Server</li><li>1: Windows Server 2003, interim level</li></ul></li></ul>



Argument	Type	Description
		2: Windows Server 2003
		3: Windows Server 2008
		4: Windows Server 2008 R2
		5: Windows Server 2012
		6: Windows Server 2012 R2
		7: Windows Server 2016, preview
		■ <b>forest_level</b> returns the functional level of the forest that contains the bound domain.

## Return value

This command returns a forest name, server name, security identifier, or functional level depending on the provided argument.

## Examples

```
get_bind_info acme.com server
```

This example returns the name of the domain controller:  
adserve02.acme.com

## Related commands

The following commands perform actions related to this command:

- **bind** binds ADEdit to a domain for subsequent ADEdit commands.
- **pop** restores the context from the top of ADEdit's context stack to ADEdit.
- **push** saves ADEdit's current context to ADEdit's context stack.
- **show** returns the current context of ADEdit, including its bound domains and its currently selected objects.

.....

## get\_child\_zones

Use the `get_child_zones` command to return a Tcl list of the child zones, computer roles, and computer zones for the currently selected zone stored in memory. The options to return child zones and computer roles are only applicable when you are working with hierarchical zones.

In classic zones, you can use this command to return a Tcl list of classic-computer zones under the currently selected classic zone. A classic-computer zone is a special zone type that contains a single computer to enable computer-level role assignments. The classic zone must have the corresponding computer object and that computer must be identified as a classic-computer zone to support computer-specific role assignments.

Because classic zones do not have child zones or computer roles, executing `get_child_zones` with the `-crole` or `-tree` option without the `-computer` option returns an empty list.

## Zone type

Classic and hierarchical

## Syntax

```
get_child_zones [-tree] [-crole] [-computer]
```

## Abbreviation

gcz

## Options

This command takes any of the following options:



Option	Description
-tree	Returns a Tcl list of the current zone's child zones. If the currently selected zone is a classic zone, this option is ignored.
-crole	Returns a Tcl list of the current zone's hosted computer roles. If the currently selected zone is a classic zone, this option is ignored.
.	Returns a Tcl list of the current zone's computer-specific zones.
computer	For classic zones, this option returns a list of classic-computer zones.

If you don't specify an option and the currently selected zone is a hierarchical zone, `get_child_zones` returns the complete list of child zones including computer roles and computer-specific "zones" that enable computer-specific overrides. If you don't specify an option and the currently selected zone is a classic zone, `get_child_zones` returns the list of classic-computer zones.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of child zones, computer roles, or computer-specific zones depending on the options used.

## Examples

```
get_child_zones
```

This example returns:

```
{CN=cz1,CN=Zones,CN=Centrify,CN=Program  
Data,DC=eel,DC=nest}  
{CN=cz2,CN=Zones,CN=Centrify,CN=Program  
Data,DC=eel,DC=nest}
```

```
{CN=global,CN=Zones,CN=Centrify,CN=ProgramData,DC=eel,DC=ne  
st/oracleServers}
```



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the zone to work with:

- `create_zone` creates a new zone in Active Directory.
- `get_zones` returns a Tcl list of all zones within a specified domain.
- `select_zone` retrieves a zone from Active Directory and stores it in memory as the currently selected zone.

After you have a zone stored in memory, you can use the following commands to work with that zone:

- `delegate_zone_right` delegates administrative rights to a specified user or group.
- `delete_zone` deletes the selected zone from Active Directory and memory.
- `get_zone_field` reads a field value from the currently selected zone.
- `get_zone_nss_vars` returns the NSS substitution variable for the selected zone.
- `save_zone` saves the selected zone with its current settings to Active Directory.
- `set_zone_field` sets a field value in the currently selected zone.

## get\_dz\_commands

Use the `get_dz_commands` command to check Active Directory and return a Tcl list of UNIX command objects defined within the currently selected zone. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use the `list_dz_commands` command to output to `stdout`.

You can only use the `get_dz_commands` command if the currently selected zone is a classic4 or hierarchical zones. The command does not work in other types of zones.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_dz_commands
```

## Abbreviation

```
gdzc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of UNIX commands defined in the currently selected zone.

## Examples

```
get_dz_commands
```

This example returns the list of commands: root\_any



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a UNIX command to work with:

- `list_dz_commands` lists to `stdout` the UNIX commands in the current zone.
- `new_dz_command` creates a new UNIX command and stores it in memory.
- `select_dz_command` retrieves a UNIX command from Active Directory and stores it in memory.

After you have a UNIX command stored in memory, you can use the following commands to work with that command:

- `delete_dz_command` deletes the selected command from Active Directory and from memory.
- `get_dzc_field` reads a field value from the currently selected command.
- `save_dz_command` saves the selected command with its current settings to Active Directory.
- `set_dzc_field` sets a field value in the currently selected command.

## `get_dzc_field`

Use the `get_dzc_field` command to return the value for a specified field from the currently selected command object that is stored in memory.

The `get_dzc_field` command does *not* query Active Directory for the command. If you change field values using ADEdit without saving the command to Active Directory, the field value you retrieve using `get_dzc_field` won't match the same field value for the command stored in Active Directory.

You can only use the `get_dzc_field` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_dzc_field field
```

## Abbreviation

gdzcf

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
field	string	<p>Required. Specifies the case-sensitive name of the field whose value to retrieve. The possible values are:</p> <ul style="list-style-type: none"> <li>▪ <b>description:</b> Returns text describing the UNIX command.</li> <li>▪ <b>cmd:</b> Returns the restricted shell command string or strings.</li> <li>▪ <b>path:</b> Returns the path to the command's location.</li> <li>▪ <b>form:</b> Returns an integer that indicates whether the cmd and path strings use wild cards (0) or a regular expression (1).</li> <li>▪ <b>dzdo_runas:</b> Returns a list of users and groups that can run this command under dzdo version of sudo. Users may be listed by user name or user ID (UID).</li> <li>▪ <b>dzsh_runas:</b> Returns a list of users and groups that can run this command in a restricted shell environment (dzsh). Users can be listed by user name or UID. You cannot get this field value if the selected zone is a classic4 zone.</li> <li>▪ <b>keep:</b> Returns a comma-separated list of environment variables from the current user's environment to keep.</li> <li>▪ <b>del:</b> Returns a comma-separated list of environment variables from the current user's environment to delete.</li> <li>▪ <b>add:</b> Returns a comma-separated list of environment variables to add to the final set of environment variables.</li> <li>▪ <b>pri:</b> Returns an integer that specifies the command priority for the restricted shell command object.</li> <li>▪ <b>umask:</b> Returns an integer that defines who can execute the command.</li> <li>▪ <b>flags:</b> Returns an integer that specifies a combination of different properties for the command.</li> </ul>

Argument	Type	Description
		<ul style="list-style-type: none"> <li>▪ <b>createTime:</b> Returns the time and date this command was created, returned in generalized time format.</li> <li>▪ <b>modifyTime:</b> Returns the time and date this command was last modified, returned in generalized time format.</li> <li>▪ <b>dn:</b> Returns the command’s distinguished name.</li> <li>▪ <b>selinux_role:</b> Returns the SELinux role used when constructing a new security context for command execution (tree zone only).</li> <li>▪ <b>selinux_type:</b> Returns the SELinux type used when constructing a new security context for command execution (tree zone only).</li> <li>▪ <b>digest:</b> Returns the SHA-2 digest to verify the file checksum before command execution.</li> </ul>
<p>Note that <code>selinux_role</code> and <code>selinux_type</code> are only supported on Red Hat Enterprise Linux systems and effective only on systems with SELinux enabled and joined to a hierarchical zone.</p>		

## Getting the cmd and path field values

If you specify the `cmd` and `path` fields, the return value can be a string that uses wild cards (\*, ?, and !), or a regular expression. If the `cmd` and `path` strings use wild cards, an asterisk (\*) matches zero or more characters, a question mark (?) matches exactly one character, and the exclamation mark (!) negates matching of the specified string.

For both the `cmd` and `path` fields, the `form` field indicates whether the specified string is interpreted as a regular expression or as a string that includes wild cards.

## Getting environment variable field values

If you specify the `keep`, `del`, or `add` field, the return value is a comma-separated list of environment variables. The `keep`, `del`, and `add` fields control



the environment variables used by the commands specified by the `cmd` string. The `keep` and `del` settings are mutually exclusive:

- The `keep` field only takes effect if the flag `16` is included in the setting for the `flag field`.
- The `del` field only takes effect if the flag `16` is not included in the setting for the `flag field`.

Any environment variables kept or deleted are in addition to the default set of the user's environment variables that are either retained or deleted. The default set of environment variables to keep is defined in the `dzdo.env_keep` configuration parameter in the `centrifdc.conf` file. The default set of environment variables to delete is defined in the `dzdo.env_delete` configuration parameter in the `centrifdc.conf` file.

The `add field` returns the environment variables added to the final set of environment variables resulting from the `keep` or `del` fields.

## Getting the command priority field value

If you specify the `pri` field, the return value indicates the command priority when there are multiple matches for command strings in a command object. If there are multiple commands specified by this command object, the `pri` field specifies their relative priority. The higher the value returned by this field, the higher the command's priority.

## Getting the umask field value

If you specify the `umask` field, the return value is a 3-digit octal value that defines who can read, write, and execute the selected command object. The three digits of the `umask` field specify the read, write, or execute permission for the file owner, group, and other users. The left digit defines the owner execution rights, the middle digit defines the group execution rights, and the right digit defines execution rights for other users. Each digit is a combination of binary flags, one flag for each right as follows:

- 4 is read
- 2 is write

• • • • •

- 1 is execute

These values are added together to define the rights available for each entity. For example, a `umask` value of 600 indicates read and write permission (4+2) for the owner, but no permissions for the group or other users. Similarly, a `umask` value of 740 indicates read, write, execute permissions (4+2+1) for the owner, read permissions for the group, but no permissions for other users.

## Getting command properties from the flags field value

If you specify the `flags` field, the return value is an integer that defines a combination of binary flags, with one flag for each of the following properties:

- 1**—Prevents nested command execution. If this flag value is not set, nested command execution is allowed.
- 2**—Requires authentication with the user's password.
- 4**—Requires authentication with the run-as user's password.
- 8**—Preserves group membership. If this flag value is not set, group membership is not preserved.
- 16**—Resets environment variables for the command, deleting the variables specified in the `dzdo.env_delete` parameter and keeping the variables specified in the `keep` field. If this flag is not set, the command removes the unsafe environment variables specified in the `dzdo.env_delete` parameter along with any additional environment variables specified by the `del` field.
- 32**—Requires multi-factor authentication to execute the command.
- 64**—Prevents navigation up the path hierarchy when executing the command.

These values are added together to define the value for the `flags` field. For example, a `flags` field value of 11 indicates that nested command execution is not allowed (1), the command requires authentication using the user's password (2), and the user's group membership should be preserved (8). The value returned is the sum of these flags (1+2+8).



## Return value

This command returns a field value, which varies in type depending on the data type stored by the field.

## Examples

```
get_dzc_field dzdo_runas
```

returns: root

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a UNIX command to work with:

- [get\\_dz\\_commands](#) returns a Tcl list of UNIX commands in the current zone.
- [list\\_dz\\_commands](#) lists to `stdout` the UNIX commands in the current zone.
- [new\\_dz\\_command](#) creates a new UNIX command and stores it in memory.
- [select\\_dz\\_command](#) retrieves a UNIX command from Active Directory and stores it in memory.

After you have a UNIX command stored in memory, you can use the following commands to work with that command:

- [delete\\_dz\\_command](#) deletes the selected command from Active Directory and from memory.
- [save\\_dz\\_command](#) saves the selected command with its current settings to Active Directory.
- [set\\_dzc\\_field](#) sets a field value in the currently selected command.



## get\_group\_members

Use the `get_group_members` command to check the Active Directory group membership for a specified group. You can use this command to return a Tcl list of the users in a specified group in one of two ways:

- With the `-ad` option to return a simplified list of the distinguished names that are members of the specified group. The `-ad` option lists the users and groups that are members of the specified group without recursively expanding the group membership of any nested group.
- Without the `-ad` option to return a complete list of users that are members of the specified group. If you don't specify the `-ad` option, the command recursively expands the groups that are members of the specified group to identify all of the users in any nested group.

## Zone type

Not applicable

## Syntax

```
get_group_members [-ad | -upn] group_UPN
```

## Abbreviation

ggm

## Options

This command takes the following options:



---

Option	Description
-ad	Returns the distinguished names for the users and groups that are members of the specified group. This option does not expand the group membership to list users who are members of nested groups.
-upn	Returns user names in user principal name (UPN) format for all of the users that are members of the specified group. This option expands the group membership of the specified group to include users who are members of nested groups.  If you don't specify this option, a complete list of user names is returned using the default <code>SAMAccountName@domain</code> format.

---

## Arguments

This command takes the following argument:

---

Argument	Type	Description
group_ UPN	string	Required. Specifies the user principal name (UPN) of the group to for which you want to return user membership.

---

## Return value

This command returns a Tcl list of group members.

## Examples

```
get_group_members poweradmins@acme.com
```

This example returns the complete list of users who are members of the `poweradmin@acme.com` group, including users who are members of any nested groups, using the `sAMAccountName@domain.name` format. For example:

```
martin.moore@acme.com rachel.roberts@acme.com  
frank.smith@acme.com
```

The following example returns the distinguished names of the users and groups that are members of the `demo-qa-lab@acme.com` group without listing the members of any nested groups.

```
get_group_members -ad demo-qa-lab@acme.com
```

.....

For example, this command returns the list of users and groups without expanding the group membership for the LabAdmins and QA groups:

```
CN=LabAdmins,CN=Users,DC=acme,DC=com {CN=Chris  
Howard,CN=Users,DC=acme,DC=com}  
CN=QA,CN=Users,DC=acme,DC=com  
CN=frank.smith,CN=UsersDC=acme,DC=com
```

## Related commands

The following commands perform actions related to this command:

- [joined\\_get\\_user\\_membership](#) checks Active Directory through `adcli` and returns a Tcl list of groups that a user belongs to.
- [joined\\_user\\_in\\_group](#) checks Active Directory through `adcli` to see if a user is in a group.
- [get\\_effective\\_groups](#) checks Active Directory and returns a Tcl list of groups a user belongs to.

## get\_local\_group\_profile\_field

Use the `get_local_group_profile_field` command to display the value of the specified field for the currently selected local UNIX or Linux group that has a profile defined in the current zone. Before executing this command, you must select a local group by executing the `select_local_group_profile` command.

## Zone type

Hierarchical only.

## Syntax

```
get_local_group_profile_field field_name
```



## Abbreviation

glgpf

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
field_name		<p>Required. Specifies the local group field to retrieve. The data type depends on the field. The possible values are:</p> <ul style="list-style-type: none"><li>▪ <b>gid</b>: The numeric group identifier.</li><li>▪ <b>name</b>: The UNIX name of the group.</li><li>▪ <b>member</b>: The UNIX name of at least one group member.</li><li>▪ <b>profileflag</b>: The value of the group's profile flag as set in the group object in the zone. For the group to be managed by the agent, the profile flag must be set to 1 or 3.  If set to 1, the group profile is enabled. If the group profile is complete and the profile flag is set to 1, the profile will be installed or updated in <code>/etc/group</code> at the next local account refresh interval.  If set to 3, the group profile is removed from <code>/etc/group</code> at the next local account refresh interval.</li><li>▪ <b>dn</b>: The distinguished name of the group.</li><li>▪ <b>createTime</b>: The creation time of the group profile.</li><li>▪ <b>modifyTime</b>: The most recent modification time of the group profile.</li></ul> <p>You can also specify AIX extended attributes as the field to get an extended attribute value for a group. Extended attribute fields start with the <code>aix.</code> prefix. For example, the <code>admin</code> extended attribute can be retrieved by specifying <code>aix.admin</code> as the field.</p>

.....

## Return value

This command returns the value of the specified field.

## Examples

The following example returns the GID of the currently selected local group in the zone.

```
get_local_group_profile_field gid
```

The following example returns the value of the profile flag for the currently selected local group. In this example, the profile flag is 1, meaning that the group profile in `/etc/group` will be updated with the latest settings from the local account zone object at the next local account refresh interval.

```
get_local_group_profile_field profileflag
```

```
1
```

If the current group is on AIX, you can get group extended attributes and values. For example, to find out if the current group is an administrative group, you can get the `admin` extended attribute:

```
get_local_group_profile_field aix.admin
```

```
true
```

## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- [delete\\_local\\_group\\_profile](#) deletes a local UNIX or Linux group that has a profile defined in the current zone.
- [delete\\_local\\_user\\_profile](#) deletes a local UNIX or Linux user that has a profile defined in the current zone.
- [get\\_local\\_groups\\_profile](#) displays a TCL list of profiles for local groups that are defined in the current zone.



- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## `get_local_groups_profile`

Use the `get_local_groups_profile` command to return a TCL list of profiles for local groups that are defined in the currently selected zone.

.....

## Zone type

Hierarchical only.

## Syntax

```
get_local_groups_profile
```

## Abbreviation

```
glgp
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

If you run this command from the command line, it returns a TCL list of profiles for local groups that are defined in the currently selected zone. The list is sorted by group UNIX name.

If you run this command in a script, no output is returned to `stdout`, and no output appears in the shell where the script is executed. To return output to `stdout` from a script, use the `list_local_groups_profile` command.



## Examples

The following example shows a TCL list of profiles for local groups that are defined in the current zone.

```
get_local_groups_profile  
lg001 lg002 lg003 lg005 lg006 lg007
```

## Related commands

The following related ADEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- [delete\\_local\\_group\\_profile](#) deletes a local UNIX or Linux group that has a profile defined in the current zone.
- [delete\\_local\\_user\\_profile](#) deletes a local UNIX or Linux user that has a profile defined in the current zone.
- [get\\_local\\_user\\_profile\\_field](#) displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- [get\\_local\\_users\\_profile](#) displays a TCL list of profiles for local users that are defined in the current zone.
- [list\\_local\\_groups\\_profile](#) displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- [list\\_local\\_users\\_profile](#) displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- [new\\_local\\_group\\_profile](#) creates an object for a local UNIX or Linux group in the currently selected zone.
- [new\\_local\\_user\\_profile](#) creates an object for a local UNIX or Linux user in the currently selected zone.
- [save\\_local\\_group\\_profile](#) saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.



- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## get\_local\_user\_profile\_field

Use the `get_local_user_profile_field` command to display the value of the specified field for the currently selected local UNIX or Linux user that has a profile defined in the current zone. Before executing this command, you must select a local user by executing the `select_local_user_profile` command.

## Zone type

Hierarchical only.

## Syntax

```
get_local_user_profile_field field_name
```

## Abbreviation

glupf



## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
field_name	string	<p>Required. Specifies the local user profile field to retrieve. The possible values include:</p> <ul style="list-style-type: none"><li>▪ <b>uid</b>: The user's numeric identifier.</li><li>▪ <b>gid</b>: The GID of the user's primary group.</li><li>▪ <b>shell</b>: The local user's default shell on the local computer. Possible values are: /bin/bash, /bin/csh, /bin/ksh, /bin/sh, /bin/tcsh, %shell}.</li><li>▪ <b>home</b>: The local user's default home directory on the local computer.</li><li>▪ <b>gecos</b>: General information about the local user account.</li><li>▪ <b>uname</b>: The UNIX name of the user.</li><li>▪ <b>dn</b>: The distinguished name of the user.</li><li>▪ <b>createTime</b>: The creation time of the user profile.</li><li>▪ <b>modifyTime</b>: The most recent modification time of the user profile.</li><li>▪ <b>profileflag</b>: The value of the user's profile flag as set in the user object in the zone. For the user to be managed by the agent, the profile flag must be set to 1, 2, or 3.</li></ul> <p>You can also specify AIX extended attributes as the field to get an extended attribute value for a user. Extended attribute fields start with the <b>aix.</b> prefix. For example, the <b>admin</b> extended attribute can be retrieved by specifying <b>aix.admin</b> as the field.</p>

.....

## Return value

This command returns the value of the specified field.

## Examples

The following example returns the UID of the currently selected local user in the zone.

```
get_local_user_profile_field uid
```

The following example returns the value of the profile flag for the currently selected local user. In this example, the profile flag is 2, meaning that the user profile in `/etc/passwd` will be updated with the latest settings from the local account zone object at the next local account refresh interval, but the password entry in `/etc/passwd` will be set to `!!` so that the user cannot log into the local computer.

```
get_local_user_profile_field profileflag  
2
```

For more information about the meaning of the profile flag value, see [set\\_local\\_user\\_profile\\_field](#).

You can also specify AIX extended attributes as the field to get an extended attribute value for a user. Extended attribute fields start with the `aix.` prefix. For example, the `admin` extended attribute can be retrieved by specifying `aix.admin` as the field.

```
get_local_user_profile_field aix.admin  
false
```

## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:



- `delete_local_group_profile` deletes a local UNIX or Linux group that has a profile defined in the current zone.
- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.

.....

- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## get\_local\_users\_profile

Use the `get_local_users_profile` command to return a TCL list of profiles for local users that are defined in the currently selected zone.

### Zone type

Hierarchical only.

### Syntax

```
get_local_users_profile  
glup
```

### Options

This command takes no options.

### Arguments

This command takes no arguments.

### Return value

If you run this command from the command line, it returns a TCL list of profiles for local users that are defined in the currently selected zone. The list is sorted by user UNIX name.



If you run this command in a script, no output is returned to `stdout`, and no output appears in the shell where the script is executed. To return output to `stdout` from a script, use the `list_local_users_profile` command.

## Examples

The following example shows a TCL list of profiles for local users that are defined in the current zone.

```
get_local_users_profile
db2011 db2012 lu001 lu002 lu003 lu004 lu006 lu007 lu008
lu009 lu012 lu013
```

## Related commands

The following related ADEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- [delete\\_local\\_group\\_profile](#) deletes a local UNIX or Linux group that has a profile defined in the current zone.
- [delete\\_local\\_user\\_profile](#) deletes a local UNIX or Linux user that has a profile defined in the current zone.
- [get\\_local\\_groups\\_profile](#) displays a TCL list of profiles for local groups that are defined in the current zone.
- [get\\_local\\_user\\_profile\\_field](#) displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- [list\\_local\\_groups\\_profile](#) displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- [list\\_local\\_users\\_profile](#) displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- [new\\_local\\_group\\_profile](#) creates an object for a local UNIX or Linux group in the currently selected zone.
- [new\\_local\\_user\\_profile](#) creates an object for a local UNIX or Linux user in the currently selected zone.



- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## get\_nis\_map

Use the `get_nis_map` command to return a Tcl list containing the entries for the currently selected NIS map stored in memory. This command does not return the contents of the comment field. If you want to retrieve the comment, use `get_nis_map_with_comment` instead.

The `get_nis_map` command does *not* query Active Directory for this NIS map, but changing map entries using `add_map_entry` and `delete_map_entry` changes both selected NIS map in memory and the corresponding NIS map in Active Directory so their contents should match.

## Zone type

Not applicable

.....

## Syntax

```
get_nis_map
```

## Abbreviation

```
gnm
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of NIS map entries. Each entry contains:

- The key
- The instance number of the key (there may be multiple entries with the same key)
- The value

Each entry component is separated from the next by a colon (:).

## Examples

```
get_nis_map
```

This example returns the list of map entries. For example:

```
{Finance:1: Hank@acme.com, jane@acme.com, joe@acme.com}  
{Mktg:1: Mike@acme.com, Sue@acme.com}
```



## Related commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and manage NIS maps:

- `delete_nis_map` deletes the selected NIS map from Active Directory and from memory.
- `get_nis_maps` returns a Tcl list of NIS maps in the currently selected zone.
- `list_nis_maps` lists to `stdout` all NIS maps in the currently selected zone.
- `new_nis_map` creates a new NIS map and stores it in memory.
- `save_nis_map` saves the selected NIS map with its current entries to Active Directory.
- `select_nis_map` retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map's entries:

- `add_map_entry` or `add_map_entry_with_comment` adds an entry to the currently selected NIS map.
- `delete_map_entry` removes an entry from the currently selected NIS map.
- `get_nis_map_with_comment` returns a Tcl list of the entries in the currently selected NIS map.
- `get_nis_map_field` reads a field value from the currently selected NIS map.
- `list_nis_map` or `list_nis_map_with_comment` lists to `stdout` of the entries in the currently selected NIS map.

## `get_nis_map_field`

Use the `get_nis_map_field` command to return the value for a specified field from the currently selected NIS map stored in memory. The `get_nis_map_field` command does *not* query Active Directory for the NIS map. If you've



changed field values using ADEdit without saving the NIS map to Active Directory, the field value you retrieve using `get_nis_map_field` won't match the same field value for the NIS map stored in Active Directory.

## Zone type

Not applicable

## Syntax

```
get_nis_map_field field
```

## Abbreviation

gnmf

## Options

This command takes no options.

## Arguments

This command takes the following argument, which is case-sensitive:

Argument	Type	Description
field	string	<p>Required. Specifies the case-sensitive name of the field whose value to retrieve. The possible values are:</p> <ul style="list-style-type: none"><li>▪ <b>createTime</b>: Specifies the time and date this NIS map was created, returned in generalized time format</li><li>▪ <b>modifyTime</b>: Specifies the time and date this NIS map was last modified, returned in generalized time format</li><li>▪ <b>dn</b>: Specifies the NIS map's distinguished name</li></ul>



## Return value

This command returns a field value, which varies in type depending on the data type stored by the field.

## Examples

```
get_nis_map_field createTime
```

This example returns the value of the `createTime` field. For example:  
20110525163718.0Z

## Related Commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and manage NIS maps:

- [delete\\_nis\\_map](#) deletes the selected NIS map from Active Directory and from memory.
- [get\\_nis\\_maps](#) returns a Tcl list of NIS maps in the currently selected zone.
- [list\\_nis\\_maps](#) lists to `stdout` all NIS maps in the currently selected zone.
- [new\\_nis\\_map](#) creates a new NIS map and stores it in memory.
- [save\\_nis\\_map](#) saves the selected NIS map with its current entries to Active Directory.
- [select\\_nis\\_map](#) retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map's entries:

- [add\\_map\\_entry](#) or [add\\_map\\_entry\\_with\\_comment](#) adds an entry to the currently selected NIS map.
- [delete\\_map\\_entry](#) removes an entry from the currently selected NIS map.



- `get_nis_maps` or `get_nis_map_with_comment` returns a Tcl list of NIS maps in the currently selected zone.
- `list_nis_map` or `list_nis_map_with_comment` lists to `stdout` of the entries in the currently selected NIS map.

## get\_nis\_map\_with\_comment

Use the `get_nis_map` command to return a Tcl list containing the entries for the currently selected NIS map stored in memory. This command includes the comment field for map entries. The `get_nis_map_with_comment` command does *not* query Active Directory for this NIS map, but changing map entries using `add_map_entry` and `delete_map_entry` changes both selected NIS map in memory and the corresponding NIS map in Active Directory so their contents should match.

## Zone type

Not applicable

## Syntax

```
get_nis_map_with_command
```

## Abbreviation

gnmwc

## Options

This command takes no options.

.....

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of NIS map entries. Each entry contains:

- The key
- The instance number of the key (there may be multiple entries with the same key)
- The value
- The comment

Each entry component is separated from the next by a colon (:).

## Examples

```
get_nis_map_with_comment
```

This example returns the map entries including comments:

```
{Finance:1: Hank@acme.com,jane@acme.com,joe@acme.com:  
Finance dept staff}  
{Mktg:1: Mike@acme.com,Sue@acme.com: Marketing dept staff}
```

## Related commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and manage NIS maps:

- [delete\\_nis\\_map](#) deletes the selected NIS map from Active Directory and from memory.
- [get\\_nis\\_maps](#) returns a Tcl list of NIS maps in the currently selected zone.
- [list\\_nis\\_maps](#) lists to stdout all NIS maps in the currently selected zone.

• • • • •

- `new_nis_map` creates a new NIS map and stores it in memory.
- `save_nis_map` saves the selected NIS map with its current entries to Active Directory.
- `select_nis_map` retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map's entries:

- `add_map_entry` or `add_map_entry_with_comment` adds an entry to the currently selected NIS map.
- `delete_map_entry` removes an entry from the currently selected NIS map.
- `get_nis_map_field` reads a field value from the currently selected NIS map.
- `get_nis_maps` returns a Tcl list of NIS maps in the currently selected zone.
- `list_nis_map` or `list_nis_map_with_comment` lists to `stdout` of the entries in the currently selected NIS map.

## get\_nis\_maps

Use the `get_nis_maps` command to check Active Directory and return a Tcl list of NIS maps defined within the currently selected zone. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use `list_nis_maps` to output the list of NIS maps to `stdout`.

## Zone type

Not applicable

## Syntax

```
get_nis_maps
```

.....

## Abbreviation

gnms

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of NIS maps defined in the currently selected zone.

## Examples

```
get_nis_maps
```

This example returns the list of NIS maps: `Aliases Printers Services`

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and manage NIS maps:

- `delete_nis_map` deletes the selected NIS map from Active Directory and from memory.
- `list_nis_maps` lists to `stdout` all NIS maps in the currently selected zone.
- `new_nis_map` creates a new NIS map and stores it in memory.



- `save_nis_map` saves the selected NIS map with its current entries to Active Directory.
- `select_nis_map` retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the other commands to work with that map's entries.

## get\_object\_field

Use the `get_object_field` command to return the value of a specified field from the currently selected Active Directory object stored in memory. The `get_object_field` command does *not* query Active Directory for the object. If you change field values using ADEdit without saving the object to Active Directory, the field value you retrieve using `get_object_field` won't match the same field value for the object stored in Active Directory.

## Zone type

Not applicable

## Syntax

```
get_object_field field
```

## Abbreviation

gof

## Options

This command takes no options.



## Arguments

This command takes the following argument:

Argument	Type	Description
field	string	<p>Required. Specifies the case-sensitive name of the field whose value to retrieve. The possible values include any attribute that can be defined for the type of object currently selected. Special values are:</p> <ul style="list-style-type: none"><li>▪ <b>sid</b>: The object's security identifier.</li><li>▪ <b>guid</b>: The object's globally unique identifier.</li><li>▪ <b>sd</b>: The object's security descriptor.</li><li>▪ <b>createTime</b>: The time and date this object was created, returned in generalized time format.</li><li>▪ <b>modifyTime</b>: The time and date this object was last modified, returned in generalized time format.</li><li>▪ <b>dn</b>: The object's distinguished name.</li></ul>

## Return value

This command returns a field value, which varies in type depending on the data type stored by the field.

## Examples

```
get_object_field guid
```

This example returns the globally unique identifier for an object. For example:

```
44918ee7-80bc-4741-95d3-dd189e235ab8
```

## Related commands

Before you use this command, you must have a currently selected Active Directory object stored in memory. The following commands enable you to view and select the object to work with:



- `get_objects` performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of matching objects.
- `new_object` creates a new Active Directory object and stores it in memory.
- `select_object` retrieves an object with its attributes from Active Directory and stores it in memory.

After you have an Active Directory object stored in memory, you can use the following commands to work with that object's attributes, delete the object, or save information for the object:

- `add_object_value` adds a value to a multi-valued field attribute of the currently selected Active Directory object.
- `delete_object` deletes the selected Active Directory object from Active Directory and from memory.
- `delete_sub_tree` deletes an Active Directory object and all of its children from Active Directory.
- `get_object_field_names` returns a Tcl list of the field names (attributes) for the currently selected Active Directory object.
- `remove_object_value` removes a value from a multi-valued field attribute of the currently selected Active Directory object.
- `save_object` saves the selected Active Directory object with its current settings to Active Directory.
- `set_object_field` sets a field value in the currently selected Active Directory object.

## `get_object_field_names`

Use the `get_object_field_names` command to return a Tcl list of the field names for each of the fields—the object attributes—of the currently selected Active Directory object. The `get_object_field_names` command does not query Active Directory for the object's field names but looks at the selected object as it is stored in ADEdit memory.

.....

## Zone type

Not applicable

## Syntax

```
get_object_field_names
```

## Abbreviation

```
gofn
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of field names.

## Examples

```
select_object "cn=amy adams,cn=users,dc=ajax,dc=com"  
get_object_field_names
```

This example returns the field names associated with the selected user Amy Adams:

```
_SID _dn _objectCategory _server accountExpires cn codePage  
countryCode distinguishedName
```

.....

```
gidNumber instanceType lastLogonTimestamp loginShell  
msDS-MembersForAzRoleBL msSFU30NisDomain  
nTSecurityDescriptor name objectCategory objectClass  
objectGUID objectSid primaryGroupID  
pwdLastSet sAMAccountName sAMAccountType uSNChanged  
uSNCreated uid uidNumber unixHomeDirectory  
userAccountControl userPrincipalName whenChanged  
whenCreated
```

## Related commands

Before you use this command, you must have a currently selected Active Directory object stored in memory. The following commands enable you to view and select the object to work with:

- [get\\_objects](#) performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of objects that match the search criteria.
- [new\\_object](#) creates a new Active Directory object and stores it in memory.
- [select\\_object](#) retrieves an object and its attributes from Active Directory and stores it in memory.

After you have an Active Directory object stored in memory, you can use the following commands to work with that object's attributes, delete the object, or save information for the object:

- [add\\_object\\_value](#) adds a value to a multi-valued field attribute of the currently selected Active Directory object.
- [delete\\_object](#) deletes the selected Active Directory object from Active Directory and from memory.
- [delete\\_sub\\_tree](#) deletes an Active Directory object and all of its children from Active Directory.
- [get\\_object\\_field](#) reads a field value from the currently selected Active Directory object.
- [remove\\_object\\_value](#) removes a value from a multi-valued field attribute of the currently selected Active Directory object.



- `save_object` saves the selected Active Directory object with its current settings to Active Directory.
- `set_object_field` sets a field value in the currently selected Active Directory object.

## get\_objects

Use the `get_objects` command to perform an LDAP search of Active Directory and return a Tcl list of the distinguished names (DNs) of the objects that match the search criteria. You specify a container in Active Directory where the search begins and a standard LDAP filter that defines the objects you're searching for.

You can control the nature of the search through options that specify whether to use the global catalog (GC) for a forest-wide search, the number of levels deep for the search to go below the beginning container of the search, and the maximum number of objects for the `get_objects` command to return.

## Zone type

Not applicable

## Syntax

```
get_objects [-gc] [-depth one|sub] [-limit limit] [-f  
forest] base filter
```

## Abbreviation

go

## Options

This command takes the following options:



Option	Description
<code>-gc</code>	Requests a forest-wide search using a global catalog. For this option to work, ADEdit must be bound to a global catalog domain controller using the <code>bind</code> command with the <code>-gc</code> option.  If you don't specify this option, the search is only within the currently bound domains.
<code>-depth one   sub</code>	Specifies how deep to search. This option must be followed by one of two values: <ul style="list-style-type: none"><li>▪ <b>one</b>: Specifies that the search will search only through objects immediately below the container specified by the argument <i>base</i>.</li><li>▪ <b>sub</b>: Specifies that the search will be full-depth, starting at the container specified by <i>base</i> and continuing through all sub-containers below that level.</li></ul> If you don't specify this option, the search defaults to the value <b>one</b> .
<code>-limit limit</code>	Limits the number of objects returned by the search to the positive integer specified by <i>limit</i> .  If you don't specify this option, the search returns all matching objects without limit.
<code>-f forest</code>	Specifies the forest to search.  If you bind ADEdit to multiple forests, you can use this option to identify a specific forest to search for objects matching the criteria you specify.

## Arguments

This command takes the following arguments:

Argument	Type	Description
<code>base</code>	DN	Required. Specifies the distinguished name of an Active Directory container in which to start the search.  If you want to perform a forest-wide search using the global catalog option but do not specify the forest to search, use an empty string as the <b>base</b> argument. For example:  <code>get_objects -gc -depth sub "" (cn=demo)</code>  You should not use an empty string as the starting point for a search if you bind to multiple forests. If you bind to multiple forests, you should always specify the forest to search.
<code>filter</code>	LDAP filter	Required. A string that uses standard LDAP filter syntax to specify criteria for the search.

.....

## Return value

This command returns a Tcl list of the distinguished names of the objects matching the search criteria.

## Examples

```
get_objects "cn=users,dc=acme,dc=com" (objectclass=*)
```

This example returns a list of distinguished name matching the `objectclass` filter:

```
CN=Builtin,DC=acme,DC=com CN=Computers,DC=acme,DC=com  
{OU=Domain Controllers,DC=acme,DC=com}  
CN=ForeignSecurityPrincipals,DC=acme,DC=com  
CN=Infrastructure,DC=acme,DC=com  
CN=LostAndFound,DC=acme,DC=com  
{CN=NTDS Quotas,DC=acme,DC=com}  
{CN=Program Data,DC=acme,DC=com} CN=System,DC=acme,DC=com  
CN=Users,DC=acme,DC=com
```

## Related commands

The following commands enable you to view and select the object to work with:

- [new\\_object](#) creates a new Active Directory object and stores it in memory.
- [select\\_object](#) retrieves an object and its attributes from Active Directory and stores it in memory.

After you have an Active Directory object stored in memory, you can use the following commands to work with that object's attributes, delete the object, or save information for the object:

- [add\\_object\\_value](#) adds a value to a multi-valued field attribute of the currently selected Active Directory object.
- [delete\\_object](#) deletes the selected Active Directory object from Active Directory and from memory.



- `delete_sub_tree` deletes an Active Directory object and all of its children from Active Directory.
- `get_object_field` reads a field value from the currently selected Active Directory object.
- `remove_object_value` removes a value from a multi-valued field attribute of the currently selected Active Directory object.
- `save_object` saves the selected Active Directory object with its current settings to Active Directory.
- `set_object_field` sets a field value in the currently selected Active Directory object.

## get\_pam\_apps

Use the `get_pam_apps` command to check Active Directory and return a Tcl list of plug-in authentication module (PAM) applications defined within the currently selected zone. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use `list_pam_apps` to output the list of PAM applications to `stdout`.

You can only use the `get_pam_apps` command to return information about PAM applications if the currently selected zone is a classic4 or hierarchical zones. The command does not work for other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
get_pam_apps
```

## Abbreviation

```
gpam
```

.....

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of PAM applications defined in the currently selected zone. Each element in the string is the name of a PAM application.

## Examples

```
get_pam_apps
```

This example returns all of the PAM application rights for the selected zone:

```
dzssh-all dzssh-direct-tcpip dzssh-exec dzssh-scp dzssh-  
sftp dzssh-shell dzssh-subsystem dzssh-tcpip-forward dzssh-  
tunnel dzssh-x11-forwarding login-all ssh sshd
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. After you have a zone stored in memory, you can use the following commands to view and select the PAM application to work with:

- `list_pam_apps` lists to stdout the PAM application rights in the current zone.
- `new_pam_app` creates a new PAM application right and stores it in memory.
- `select_pam_app` retrieves a PAM application from Active Directory and stores it in memory.



After you have a PAM application stored in memory, you can use the following commands to work with that PAM application's attributes, delete the PAM application, or save information for the PAM application:

- `delete_pam_app` deletes the selected PAM application from Active Directory and from memory.
- `get_pam_field` reads a field value from the currently selected PAM application.
- `save_pam_app` saves the selected PAM application with its current settings to Active Directory.
- `set_pam_field` sets a field value in the currently selected PAM application.

## get\_pam\_field

Use the `get_pam_field` command to return the value of a specified field for the currently selected plug-in authentication module (PAM) application object stored in memory. The `get_pam_field` command does *not* query Active Directory for the PAM application. If you change field values using ADEdit without saving the PAM application to Active Directory, the field value you retrieve using `get_pam_field` won't match the same field value for the PAM application stored in Active Directory.

You can only use the `get_pam_field` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
get_pam_field field
```

.....

## Abbreviation

gp f

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
		Required. Specifies the case-sensitive name of the field whose value to retrieve. The possible values are: <ul style="list-style-type: none"><li>▪ <b>application:</b> The name of the application allowed to use adclient's PAM authentication service. The name can be literal, or it can contain ? or * wildcard characters to specify multiple applications.</li></ul>
field	string	<ul style="list-style-type: none"><li>▪ <b>description:</b> Text describing the PAM application.</li><li>▪ <b>createTime:</b> The time and date this PAM application was created, returned in generalized time format.</li><li>▪ <b>modifyTime:</b> The time and date this PAM application was last modified, returned in generalized time format.</li><li>▪ <b>dn:</b> the PAM application's distinguished name.</li></ul>

## Return value

This command returns a field value. The data type for this value depends on the field specified.



## Examples

```
get_pam_field application
```

This example returns the contents of the `application` field:

```
ftp
```

The selected PAM application object specifies `ftp` can authenticate using `adclient`.

## Related commands

Before you use this command, you must have a currently selected PAM application object stored in memory. The following commands to view and select the PAM application to work with:

- `get_pam_apps` returns a Tcl list of PAM application rights in the current zone.
- `list_pam_apps` lists to `stdout` the PAM application rights in the current zone.
- `new_pam_app` creates a new PAM application right and stores it in memory.
- `select_pam_app` retrieves a PAM application right from Active Directory and stores it in memory.

After you have a PAM application stored in memory, you can use the following commands to work with that PAM application's attributes, delete the PAM application, or save information for the PAM application:

- `delete_pam_app` deletes the selected PAM application right from Active Directory and from memory.
- `get_pam_field` reads a field value from the currently selected PAM application right.
- `save_pam_app` saves the selected PAM application right with its current settings to Active Directory.
- `set_pam_field` sets a field value in the currently selected PAM application right.

.....

## get\_parent\_dn

Use the `get_parent_dn` command to specify an LDAP path using a distinguished name (DN) and return the parent of the path. This command removes the first element from the distinguished name and returns the rest of the DN.

### Zone type

Not applicable

### Syntax

```
get_parent_dn DN
```

### Abbreviation

gpd

### Options

This command takes no options.

### Arguments

This command takes the following argument:

Argument	Type	Description
DN	string	Required. Specifies a distinguished name.

.....

## Return value

This command returns a distinguished name that is the parent of the supplied distinguished name.

## Examples

```
get_parent_dn CN=global,CN=Zones,CN=Centrify,DC=acme,DC=com
```

This example returns: CN=Zones,CN=Centrify,DC=acme,DC=com

## Related commands

The following command performs actions related to this command:

- `get_rdn` returns the relative distinguished name of a specified LDAP path.

## get\_pending\_zone\_groups

Use the `get_pending_zone_groups` command to check Active Directory and return a Tcl list of pending import groups for the currently selected zone. Pending import groups are group profiles that have been imported from Linux or UNIX computers, but not yet mapped to any Active Directory group. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use `list_pending_zone_groups` to output the list to `stdout`.

## Zone type

Classic and hierarchical

## Syntax

```
get_pending_zone_groups
```

.....

## Abbreviation

gpzg

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of pending import group profiles that have been imported into the currently selected zone. Each entry in the list contains the following fields, separated by colons (:):

- Distinguished name (DN) of the pending import group as it is stored in Active Directory. The distinguished name for each pending import group includes a prefix that consists of "PendingGroup" and the globally unique identifier (GUID) for the group.
- UNIX group name.
- Numeric group identifier (GID).

## Examples

```
get_pending_zone_groups
```

The command returns output in the form of:

```
DN:group_name:gid
```

This sample command might return output similar to the following:

```
CN=PendingGroup_573135e7-edd9-46b9-9cbd-  
c839570a90c8,CN=Groups, CN=bean_
```

.....

```
pz,CN=Zones,CN=Centrify,DC=win2k3,DC=test:root:0  
CN=PendingGroup_7878065a-4d2f-4749-8f3b-  
6ffe24303f6a,CN=Groups, CN=bean_  
pz,CN=Zones,CN=Centrify,DC=win2k3,DC=test:unixgrp:5000
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following command performs actions related to this command:

- `select_object` retrieves the specified Active Directory object and its attributes from Active Directory and stores the object in memory.
- `get_object_field` enables you to view and work with the pending import group.

## get\_pending\_zone\_users

Use the `get_pending_zone_users` command to check Active Directory and return a Tcl list of pending import users for the currently selected zone. Pending import users are user profiles that have been imported from Linux or UNIX computers, but not yet mapped to any Active Directory user. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use `list_pending_zone_users` to output the list to `stdout`.

## Zone type

Classic and hierarchical

## Syntax

```
get_pending_zone_users
```



## Abbreviation

`gpzu`

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of pending import user profiles that have been imported into the currently selected zone. Each entry in the list contains the following fields, separated by colons (:):

- Distinguished name (DN) of the pending import user as it is stored in Active Directory. The distinguished name for each pending import user includes a prefix that consists of "PendingUser" and the globally unique identifier (GUID) for the user.
- UNIX user name.
- Numeric user identifier (UID).
- Numeric primary group identifier (GID).
- Personal information from the GECOS field.
- Home directory.
- Default login shell.

## Examples

```
get_pending_zone_users
```

This sample command might return output similar to the following:

.....

```
CN=PendingUser_09024f3a-6abc-4666-a127-722f9fe0e0bf,CN=Users,CN=finance,  
CN=Zones,CN=Centrify,DC=win2k3,DC=test:root:0:0:root:/root:  
/bin/bash  
CN=PendingUser_0b9fe038-1325-438f-8529-cb190ab5914a,CN=Users,CN=finance,  
CN=Zones,CN=Centrify,DC=win2k3,DC=test:bean:6001:5000:bean.  
zhang:/home/bean:/bin/bash
```

Before you use this command, you must have a currently selected zone stored in memory. The following command performs actions related to this command:

- `select_object` retrieves the specified Active Directory object and its attributes from Active Directory and stores the object in memory.
- `get_object_field` enables you to view and work with the pending import group.

## get\_pwnam

Use the `get_pwnam` command to look up a UNIX user name in the `/etc/passwd` file on the ADEdit host computer. If there's an entry for the specified user name, the command returns the profile values of that entry as a Tcl list. The `get_pwnam` command uses the NSS layer to perform the lookup operation. You can use the command to look up information for any user in the `/etc/passwd` file, including root.

## Zone type

Not applicable

## Syntax

```
get_pwnam unix_name
```

.....

## Abbreviation

gpn

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
unix_name	string	Required. Specifies the UNIX user name to search for in the <code>/etc/passwd</code> file.

## Return value

This command returns a Tcl list of user profile attributes for a specified user if the specified user name is found in the local `/etc/passwd` file. If the command doesn't find the specified user, it a "User not found" message.

## Examples

```
get_pwnam adam
```

This example returns the profile for the UNIX user adam:

```
adam x 500 500 {Adam Andrews} /home/adam /bin/bash
```

## Related commands

The following command performs actions related to this command:



- `getent_passwd` returns a Tcl list of all entries in the local `/etc/passwd` file.

## get\_rdn

Use the `get_rdn` command to specify an LDAP path using a distinguished name (DN) and return the relative distinguished name. This command returns only the first element of the supplied distinguished name.

## Zone type

Not applicable

## Syntax

```
get_rdn DN
```

## Abbreviation

grdn

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
DN	string	Required. Specifies a distinguished name.

• • • • •

## Return value

This command returns the first element of the supplied distinguished name.

## Examples

```
get_rdn CN=global,CN=Zones,CN=Centrify,DC=acme,DC=com
```

This example returns: `CN=global`

## Related commands

The following command performs actions related to this command:

- `get_parent_dn` returns the parent distinguished name of a specified LDAP path.

## get\_role\_apps

Use the `get_role_apps` command to return a Tcl list of PAM application rights associated with the currently selected role.

The `get_role_apps` command does *not* query Active Directory for the role. If you change the PAM applications associated with the current role using `ADEdit` without saving the role to Active Directory, the PAM applications you retrieve using `get_role_apps` won't match the same PAM applications for the role as stored in Active Directory.

You can only use the `get_role_apps` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

.....

## Syntax

```
get_role_apps
```

## Abbreviation

```
grap
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of PAM applications associated with the currently selected role. Each PAM application in the list shows the application name followed by a slash (/) and the zone in which the PAM application is defined.

## Examples

```
get_role_apps
```

This example returns the list of PAM applications for the currently selected role: ftp/cz1



## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands to view and select the role to work with:

- `get_roles` returns a Tcl list of roles in the currently selected zone.
- `list_roles` lists to stdout the roles in the currently selected zone.
- `new_role` creates a new role and stores it in memory.
- `select_role` retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with that role's attributes, delete the role, or save information for the role:

- `add_command_to_role` adds a UNIX command to the currently selected role.
- `add_pamapp_to_role` adds a PAM application to the currently selected role.
- `delete_role` deletes the selected role from Active Directory and from memory.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the currently selected role.
- `get_role_field` reads a field value from the currently selected role.
- `list_role_rights` returns a list of all UNIX commands and PAM applications associated with the currently selected role.
- `remove_command_from_role` removes a UNIX command from the currently selected role.
- `remove_pamapp_from_role` removes a PAM application from the currently selected role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the currently selected role.



## get\_role\_assignment\_field

Use the `get_role_assignment_field` command to return the value for a specified field from the currently selected role assignment stored in memory. The `get_role_assignment_field` command does *not* query Active Directory for the role assignment. If you change field values using AEdit without saving the role assignment to Active Directory, the field value you retrieve using `get_role_assignment_field` won't match the same field value for the role assignment stored in Active Directory.

You can only use the `get_role_assignment_field` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

### Zone type

Classic and hierarchical

### Syntax

```
get_role_assignment_field field
```

### Abbreviation

`graf`

### Options

This command takes no options.

### Arguments

This command takes the following argument:



Argument	Type	Description
field	string	Required. Specifies the case-sensitive name of the field whose value to retrieve. The possible values are:
		<b>assignee</b> : Returns user display name in format specific to type of logged in user.
		<b>customAttr</b> : Returns the custom text strings set for the role assignment.
		<b>customAttr</b> : Returns the custom text strings set for the role assignment.
		<b>description</b> : Returns the description for the role assignment.
		<b>dn</b> : Returns the role assignment's distinguished name.
		<b>from</b> : Returns the starting date and time for the role assignment.
		The start and end dates and times are expressed in standard UNIX time. You can use the Tcl clock command to manipulate these values. A value of 0 indicates no date or time is set for the role assignment.
		<b>modifyTime</b> : Returns the time and date this role assignment was last modified, returned in generalized time format.
		<b>ptype</b> : Returns a letter or symbol that indicates the account type associated with a role assignment. You can use the explain_ptype command to translate the returned value into a text string that describes the account type.
		<b>role</b> : Returns the name of the role and the zone in which the role is defined.
		<b>to</b> : Returns the ending date and time for the role assignment.

## Return value

This command returns a field value. The data type depends on the field specified.

## Examples

This example returns the role name (root) and the zone where the role is defined (global):

```
get_role_assignment_field role
root/global
```

This example returns the assignee display name in the appropriate format.

```
get_role_assignment_field assignee
```



- For AD user/group:  
CN=dc1,CN=Users,DC=sayms,DC=local
- For trusted forest AD user/group:  
CN=S-1-5-21-4259971489-770964042-439865176-1106,CN=ForeignSecurityPrincipals,DC=sayms,DC=local
- For local uid:  
#56789@localhost
- For local user:  
localuser1@localhost
- For local group:  
%localgroup1@localhost

## Related commands

Before you use this command, you must have a currently selected role assignment stored in memory. The following commands to view and select the role assignment to work with:

### get\_role\_assignments

Use the `get_role_assignments` command to check Active Directory and return a Tcl list of role assignments defined within the currently selected zone. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use `list_role_assignments` to output the list to `stdout`.

If you do not specify an option, the command returns the current users and groups in the zone with a role assignment.

You can only use the `get_role_assignments` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_role_assignments [-upn] [-user] [-group] [-invalid]
```

## Abbreviation

gra

## Options

This command takes any one of the following options:

Option	Description
-upn	Returns user names in user principal name (UPN) format, not the default SAMAccount@domain format.
-user	Returns a Tcl list of the current users in the zone with a role assignment.
-group	Returns a Tcl list of the current groups in the zone with a role assignment.
-	Returns a Tcl list of any invalid role assignments in the zone.
invalid	For example, this option would return role assignment for a group or user that no longer exists.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of role assignments defined in the currently selected zone. Each role assignment includes the SAMAccount@domain name

• • • • •

or the user principal name of the user or group to whom the role is assigned, the name of the role assigned, and the zone in which the role is defined. These three pieces of data are separated from each other by a slash (/).

## Examples

```
get_role_assignments
```

This example returns the list of role assignments:

```
poweradmins@acme.com/root/global proj_
admins@acme.com/login/global
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. After you have a zone stored in memory, you can use the following commands to view and select the role assignment to work with:

- `list_role_assignments` lists to `stdout` the role assignments in the current zone.
- `new_role_assignment` creates a new role assignment and stores it in memory.
- `select_role_assignment` retrieves a role assignment from Active Directory and stores it in memory.

After you have a role assignment stored in memory, you can use the following commands to work with that role assignment's attributes, delete the role assignment, or save information for the role assignment:

- `delete_role_assignment` deletes the selected role assignment from Active Directory and from memory.
- `get_role_assignment_field` reads a field value from the currently selected role assignment.
- `save_role_assignment` saves the selected role assignment with its current settings to Active Directory.
- `set_role_assignment_field` sets a field value in the currently selected role assignment.

.....

## get\_role\_commands

Use the `get_role_commands` command to return a Tcl list of UNIX commands associated with the currently selected role. The `get_role_commands` command does *not* query Active Directory for the role. If you change commands associated with the current role using ADEdit without saving the role to Active Directory, the commands you retrieve using `get_role_commands` won't match the same commands for the role stored in Active Directory.

You can only use the `get_role_commands` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

### Zone type

Classic and hierarchical

### Syntax

```
get_role_commands
```

### Abbreviation

grc

### Options

This command takes no options.

### Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of commands associated with the currently selected role. Each command in the list shows the command name followed by a slash (/) and the zone in which the command is defined.

## Examples

```
get_role_commands
```

This example returns the list of commands:

```
pwd/global ls/global cd/childzone1
```

### Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with:

- [get\\_roles](#) returns a Tcl list of roles in the current zone.
- [list\\_roles](#) lists to stdout the roles in the current zone.
- [new\\_role](#) creates a new role and stores it in memory.
- [select\\_role](#) retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with that role's attributes, delete the role, or save information for the role:

- [add\\_command\\_to\\_role](#) adds a UNIX command to the currently selected role.
- [add\\_pamapp\\_to\\_role](#) adds a PAM application to the currently selected role.
- [delete\\_role](#) deletes the selected role from Active Directory and from memory.
- [get\\_role\\_apps](#) returns a Tcl list of the PAM applications associated with the currently selected role.
- [get\\_role\\_field](#) reads a field value from the currently selected role.



- `list_role_rights` returns a list of all UNIX commands and PAM applications associated with the currently selected role.
- `remove_command_from_role` removes a UNIX command from the currently selected role.
- `remove_pamapp_from_role` removes a PAM application from the currently selected role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the currently selected role.

## get\_role\_field

Use the `get_role_field` command to return the value for a specified field from the currently selected role stored in memory. The `get_role_field` command does *not* query Active Directory for the role. If you change field values using ADEdit without saving the role to Active Directory, the field value you retrieve using `get_role_field` won't match the same field value for the role stored in Active Directory.

You can only use the `get_role_field` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
get_role_field field
```

## Abbreviation

grf



## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
field	string	Required. Specifies the case-sensitive name of the field whose value to retrieve.

The possible field values are:

- **allowLocalUser**: Returns true or false depending on whether local users can be assigned to the role. You cannot get this field value if the selected zone is a classic4 zone.
- **AlwaysPermitLogin**: Returns true or false depending on whether “rescue rights” are configured for the role. You cannot get this field value if the selected zone is a classic zone.
- **auditLevel**: Returns the auditing level configured for the role. Roles can be configured without auditing (not requested), to audit if possible, or to have auditing required. You cannot get this field value if the selected zone is a classic4 zone.
- **createTime**: Returns the time and date this role was created in generalized time format.
- **customAttr**: Returns the custom text strings set for the role.
- **description**: Returns the text string that describes the role.
- **dn**: Returns the role’s distinguished name.
- **modifyTime**: Returns the time and date this role was last modified in generalized time format.
- **sysrights**: Returns the system rights granted to the role. This value is an integer that represents a combination of binary flags, one for each system right. You cannot get this field value if the selected zone is a classic zone.



For more information about the value returned for system rights, see [Getting the system rights field for a role](#).

- **timebox:** Returns the hours and days in the week when the role is enabled. This value is a 42-digit hexadecimal number.

When represented in binary, each bit represents an hour of the week as described in the [Timebox value format](#)

- **visible:** Returns true or false depending on whether “User is visible” right is configured for the role. You cannot get this field value if the selected zone is a classic zone.

## Getting the system rights field for a role

You can specify the `sysrights` field to return information about the system rights that have been granted to the currently selected role. This field value is an integer that represents a combination of binary flags, with one flag for each of the following system rights:

- 1**—Password login and non password (SSO) login are allowed.
- 2**—Non password (SSO) login is allowed.
- 4**—Account disabled in Active Directory can be used by sudo, cron, etc.
- 8**—Log in with non-restricted shell.
- 16**—Audit not requested/required.
- 32**—Audit required.
- 64**—Always permit to login.
- 128**—Remote login access is allowed for Windows computers.
- 256**—Console login access is allowed for Windows computers.
- 512**—Require multi-factor authentication through the Centrify connector to log on.
- 1024**—PowerShell remote access is allowed

These values are added together to define the `sysrights` field value. For example, a `sysrights` value of 6 indicates that the role is configured to allow single sign-on login and to ignore disabled accounts (2+4). A value of 11



indicates that the most common UNIX system rights are enabled (1+2+8). A value of 384 indicates that most common Windows system rights are enabled (128+256).

## Return value

This command returns a field value, which varies in type depending on the data type stored by the field.

## Examples

```
get_role_field timebox
```

This example returns the content of the `timebox` field:

```
00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0
```

This return value indicates that the role is enabled during all hours of the weekdays, but none of the weekends.

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with:

- [get\\_roles](#) returns a Tcl list of roles in the current zone.
- [list\\_roles](#) lists to stdout the roles in the currently selected zone.
- [new\\_role](#) creates a new role and stores it in memory.
- [select\\_role](#) retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with that role's attributes, delete the role, or save information for the role:



- `add_command_to_role` adds a UNIX command to the currently selected role.
- `add_pamapp_to_role` adds a PAM application to the currently selected role.
- `delete_role` deletes the selected role from Active Directory and from memory.
- `get_role_apps` returns a Tcl list of the PAM applications associated with the currently selected role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the currently selected role.
- `list_role_rights` returns a list of all UNIX commands and PAM applications associated with the currently selected role.
- `remove_command_from_role` removes a UNIX command from the currently selected role.
- `remove_pamapp_from_role` removes a PAM application from the currently selected role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the currently selected role.

## get\_role\_rs\_commands

Use the `get_role_rs_commands` command to return a Tcl list of the restricted shell commands associated with the currently selected role.

The `get_role_rs_commands` command does not query Active Directory for the restricted shell commands. If you change the restricted shell commands associated with the current role using ADEdit without saving the role to Active Directory, the commands you retrieve using `get_role_rs_commands` won't match the restricted shell commands that are stored in Active Directory.

You can only use `get_role_rs_commands` if the currently selected zone is a classic4 zone. This command does not work in other types of zones.

.....

## Zone type

Classic only

## Syntax

```
get_role_rs_commands
```

## Abbreviation

```
grrsc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of restricted shell commands associated with the currently selected role. Each restricted shell command in the list shows the restricted shell command name followed by a slash (/) and the zone in which the restricted shell command is defined.

## Examples

```
get_role_rs_commands
```

This example returns : `rse1-id2/c123 rse1-id1/c123`



## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with:

- `get_roles` returns a Tcl list of roles in the current zone.
- `list_roles` lists to `stdout` the roles in the currently selected zone.
- `new_role` creates a new role and stores it in memory.
- `select_role` retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with restricted shells:

- `get_role_rs_env` returns the restricted shell environment from the currently selected role.

## `get_role_rs_env`

Use the `get_role_rs_env` command to return the restricted shell environment from the currently selected role that is stored in memory.

The `get_role_rs_env` command does not query the data stored in Active Directory for the role. If you change the restricted shell environment in ADEdit without saving the role to Active Directory, the value you retrieve using `get_role_rs_env` won't match the same value for the role that is stored in Active Directory.

You can only use the `get_role_rs_env` command if the currently selected zone is a classic4 zone. The command does not work in other types of zones.

## Zone type

Classic only

.....

## Syntax

```
get_role_rs_env
```

## Abbreviation

```
grrse
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns the restricted shell environment of the currently selected role if it runs successfully. If the currently selected role does not require a restricted shell environment, the command returns nothing.

## Examples

```
get_role_rs_env
```

This example returns the restricted shell environment if it exists for the selected role:

```
rse1
```



## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with:

- `get_roles` returns a Tcl list of roles in the current zone.
- `list_roles` lists to `stdout` the roles in the currently selected zone.
- `new_role` creates a new role and stores it in memory.
- `select_role` retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with restricted shells:

- `list_rs_envs` lists to `stdout` the restricted shell environments.
- `new_rs_env` creates a new restricted shell environment and stores it in memory.
- `save_rs_env` saves the restricted shell environment to Active Directory.
- `select_rs_env` retrieves a restricted shell environment from Active Directory and stores it in memory.

## `get_roles`

Use the `get_roles` command to check Active Directory and return a Tcl list of roles defined within the currently selected zone. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use `list_roles` to output the list to `stdout`.

You can only use the `get_roles` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

.....

## Syntax

```
get_roles
```

## Abbreviation

```
getr
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of roles defined in the currently selected zone.

## Examples

```
get_roles
```

This example returns the list of roles:

```
{Rescue - always permit login} scp sftp listed {UNIX Login}  
{Windows Login} winscp
```

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with:



- `list_roles` lists to `stdout` the roles in the currently selected zone.
- `new_role` creates a new role and stores it in memory.
- `select_role` retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with role:

- `add_command_to_role` adds a UNIX command to the currently selected role.
- `add_pamapp_to_role` adds a PAM application to the currently selected role.
- `delete_role` deletes the selected role from Active Directory and from memory.
- `get_role_apps` returns a Tcl list of the PAM applications associated with the currently selected role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the currently selected role.
- `list_role_rights` returns a list of all UNIX commands and PAM applications associated with the currently selected role.
- `remove_command_from_role` removes a UNIX command from the currently selected role.
- `remove_pamapp_from_role` removes a PAM application from the currently selected role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the currently selected role.

## `get_rs_commands`

Use the `get_rs_commands` command to return a Tcl list of restricted shell commands that are defined for the currently selected zone. If you want to return a list of restricted shell commands to `stdout`, use the `list_rs_commands` command.

.....

## Zone type

Classic only

## Syntax

```
get_rs_commands
```

## Abbreviation

```
grsc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of restricted shell commands for the currently selected zone.

## Examples

```
get_rs_commands
```

This example returns output similar to this:

```
rse1-id1 rse1-id2 rse2-id1
```



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the restricted shell command to work with:

- `list_rs_commands` lists to `stdout` the restricted shell commands in the current zone.
- `new_rs_command` creates a new restricted shell command and stores it in memory.
- `select_rs_command` retrieves a restricted shell command from Active Directory and stores it in memory.

After you have a restricted shell command stored in memory, you can use the following commands to work with that restricted shell:

- `delete_rs_command` deletes the selected command from Active Directory and from memory.
- `get_rsc_field` reads a field value from the currently selected command.
- `save_rs_command` saves the selected command with its current settings to Active Directory.
- `set_rsc_field` sets a field value in the currently selected command.

## `get_rs_envs`

Use the `get_rs_envs` command to check Active Directory and return a list of restricted environments that are defined within the currently selected zone. If you want to return a list of restricted shell environment to `stdout`, use the `list_rs_envs` command.

## Zone type

Classic only

.....

## Syntax

```
get_rs_envs
```

## Abbreviation

```
grse
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of restricted environments in the currently selected zone.

## Examples

```
get_rs_envs  
rse1 rse2
```

This example returns the list of restricted shell environments.

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with restricted shell environments:



- `list_rs_envs` lists to stdout the restricted shell environments.
- `new_rs_env` creates a new restricted shell environment and stores it in memory.
- `select_rs_env` retrieves a restricted shell environment from Active Directory and stores it in memory.

After you have a restricted shell environment stored in memory, you can use the following commands to work with its fields:

- `delete_rs_env` deletes the current restricted shell environment from Active Directory and from memory.
- `get_rse_field` reads a field value from the current restricted shell environment.
- `save_rs_env` saves the restricted shell environment to Active Directory.
- `set_rse_field` sets a field value in the current restricted shell environment.

## get\_rsc\_field

Use the `get_rsc_field` command to return the value of a specified field value from the currently selected restricted shell command that is stored in memory. Centrifify-specific fields are similar to Active Directory attributes but are stored within the Active Directory schema.

The `get_rsc_field` command does not query Active Directory for the restricted shell command. If you change field values using ADEdit without saving the restricted shell command to Active Directory, the field value you retrieve using `get_rsc_field` won't match the value stored in Active Directory.

You can only use the `get_rsc_field` command if the currently selected zone is a classic4 zone. The command does not work in other types of zones.

## Zone type

Classic only

.....

## Syntax

```
get_rsc_field field
```

## Abbreviation

```
grscf
```

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
		Required. Specifies the name of the field whose value you want to retrieve. The possible values are:
		<ul style="list-style-type: none"> <li>■ <b>description</b>: Returns text describing the restricted shell command.</li> <li>■ <b>cmd</b>: Returns the restricted shell command string or strings.</li> <li>■ <b>path</b>: Returns the path to the command's location.</li> <li>■ <b>form</b>: Returns an integer that indicates whether the cmd and path strings use wild cards (0) or a regular expression (1).</li> <li>■ <b>dzsh_runas</b>: Returns a list of users and groups that can run this command in a restricted shell environment (dzsh). Users can be listed by user name or UID.</li> <li>■ <b>keep</b>: Returns a comma-separated list of environment variables from the current user's environment to keep.</li> </ul>
field	string	<ul style="list-style-type: none"> <li>■ <b>del</b>: Returns a comma-separated list of environment variables from the current user's environment to delete.</li> <li>■ <b>add</b>: Returns a comma-separated list of environment variables to add to the final set of environment variables.</li> <li>■ <b>pri</b>: Returns a n integer that specifies the command priority for the restricted shell command object.</li> <li>■ <b>umask</b>: Returns an integer that defines who can execute the command.</li> <li>■ <b>flags</b>: Returns an integer that specifies a combination of different properties for the command.</li> <li>■ <b>createTime</b>: The time and date this command was created, returned in generalized time format.</li> <li>■ <b>modifyTime</b>: The time and date this command was last modified, returned in generalized time format.</li> <li>■ <b>dn</b>: The command's distinguished name.</li> </ul>

.....

## Return value

This command returns a field value. The data type depends on the field specified. For more information about the field values returned by different fields, see [get\\_dzc\\_field](#).

## Examples

```
get_rsc_field description
```

This example returns the contents of the `description` field:

```
This is the RSC description
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the restricted shell command to work with:

- [get\\_rs\\_commands](#) returns a Tcl list of restricted shell commands in the current zone.
- [list\\_rs\\_commands](#) lists to `stdout` the restricted shell commands in the current zone.
- [new\\_rs\\_command](#) creates a new restricted shell command and stores it in memory.
- [select\\_rs\\_command](#) retrieves a restricted shell command from Active Directory and stores it in memory.

After you have a restricted shell command stored in memory, you can use the following commands to work with that restricted shell:

- [delete\\_rs\\_command](#) deletes the selected command from Active Directory and from memory.
- [save\\_rs\\_command](#) saves the selected command with its current settings to Active Directory.
- [set\\_rsc\\_field](#) sets a field value in the currently selected command.

.....

## get\_rse\_cmds

Use the `get_rse_cmds` command to return a Tcl list of restricted shell commands associated with the currently selected restricted shell environment.

The `get_rse_cmds` command does not query Active Directory for the restricted shell environment. If you change the restricted shell commands associated with the current restricted shell environment using ADEdit without saving the restricted shell environment to Active Directory, the commands you retrieve using `get_rse_cmds` won't match those stored in Active Directory.

You can only use the `get_rse_cmds` command if the currently selected zone is a classic4 zone. The command does not work in other types of zones.

## Zone type

Classic only

## Syntax

```
get_rse_cmds
```

## Abbreviation

```
grsec
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.



## Return value

This command returns a Tcl list of restricted shell commands associated with the currently selected restricted shell environment. Each restricted shell command in the list shows the command name followed by a slash (/) and the zone in which the command is defined.

## Examples

```
get_rse_cmds
```

The command returns the list restricted commands:

```
rse1-id2/c123 rse1-id1/c123
```

## Related commands

Before you use this command, you must have a currently selected restricted shell environment stored in memory. The following commands enable you to view and select the restricted shell environments:

- [list\\_rs\\_envs](#) lists to stdout the restricted shell environments.
- [new\\_rs\\_env](#) creates a new restricted shell environment and stores it in memory.
- [save\\_rs\\_env](#) saves the restricted shell environment to Active Directory.
- [select\\_rs\\_env](#) retrieves a restricted shell environment from Active Directory and stores it in memory.

After you have a restricted shell environment stored in memory, you can use the following command to work with its fields:

- [set\\_rse\\_field](#) sets a field value in the current restricted shell environment.



## get\_rse\_field

Use the `get_rse_field` command to return a field value from the currently selected restricted shell environment stored in memory.

The `get_rse_field` command does not query Active Directory for the restricted shell environment. If you have changed field values using `AEdit` without saving the restricted shell environment to Active Directory, the field value you retrieve using `get_rse_field` won't match the field value for the restricted shell environment that is stored in Active Directory.

You can only use the `get_rse_field` command if the currently selected zone is a classic4 zone. The command does not work in other types of zones.

## Zone type

Classic only

## Syntax

```
get_rse_field field
```

## Abbreviation

grsef

## Options

This command takes no options.

## Arguments

This command takes the following argument:



---

Argument	Type	Description
field	string	Required. Specifies the name of the field whose value to get. The only possible value is: <b>description</b> : Returns a text string describing the restricted shell environment.

---

## Return value

This command returns a field value, which varies in type depending on the data type stored by the field.

## Examples

```
get_rse_field description
```

This command returns the content of the `description` field. For example:

```
This is the restricted shell environment description
```

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with restricted shell environments:

- [get\\_rs\\_envs](#) returns a Tcl list of restricted shell environments.
- [list\\_rs\\_envs](#) lists to `stdout` the restricted shell environments.
- [new\\_rs\\_env](#) creates a new restricted shell environment and stores it in memory.
- [select\\_rs\\_env](#) retrieves a restricted shell environment from Active Directory and stores it in memory.

After you have a restricted shell environment stored in memory, you can use the following commands to work with its fields:



- `delete_rs_env` deletes the current restricted shell environment from Active Directory and from memory.
- `save_rs_env` saves the restricted shell environment to Active Directory.
- `set_rse_field` sets a field value in the current restricted shell environment.

## get\_schema\_guid

Use the `get_schema_guid` command to look up a specified class or attribute in Active Directory. If the specified object is found, the command returns the globally unique identifier (GUID) of the class or attribute.

This command is useful for setting a security descriptor (SD) at a class or attribute level.

## Zone type

Not applicable

## Syntax

```
get_schema_guid schema_name
```

## Abbreviation

gsg

## Options

This command takes no options.



## Arguments

This command takes the following argument:

Argument	Type	Description
schema_name	string	Required. Specifies the name of a class or attribute.

## Return value

This command returns the globally unique identifier (GUID) of the provided schema object (class or attribute).

## Examples

```
get_schema_guid MS-DS-Az-Role
```

This example returns the globally unique identifier of `MS-DS-Az-Role`:

```
8213eac9-9d55-44dc-925c-e9a52b927644
```

## Related commands

None.

## get\_zone\_computer\_field

Use the `get_zone_computer_field` command to return the value of a specified field from the currently selected zone computer stored in memory. The `get_zone_computer_field` command does *not* query Active Directory for the zone computer. If you change field values using `ADEdit` without saving the zone computer to Active Directory, the field value you retrieve using `get_zone_computer_field` won't match the same field value for the zone computer stored in Active Directory.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_zone_computer_field field
```

## Abbreviation

gzcf

## Options

This command takes no options.

## Arguments

This command takes the following argument:



---

Argument	Type	Description
field	string	<p>Required. Specifies the case-sensitive name of the field whose value to retrieve. The possible values are:</p> <ul style="list-style-type: none"><li>▪ <b>addn</b>: Returns the distinguished name of the Active Directory computer object for the zone computer. For example, if the computer object is created in the default Computers container, this field might return a path similar to <code>CN=firefly-sf,CN=Computers,DC=ajax,DC=org</code>.</li><li>▪ <b>agentVersion</b>: Returns the version of agent currently installed on the zone computer.</li><li>▪ <b>cpus</b>: Returns the number of CPUs in the computer.</li><li>▪ <b>createTime</b>: Returns the time and date this zone computer was created (in generalized time format).</li><li>▪ <b>dn</b>: Returns the distinguished name of the service connection point for the zone computer. If the computer is in a Services for UNIX (SFU) zone, no value is returned for this field.</li><li>▪ <b>dnsname</b>: Returns the domain name service (DNS) name of the zone computer.</li><li>▪ <b>enabled</b>: Returns 1 if the zone computer is enabled in its zone or 0 if it is not.</li><li>▪ <b>modifyTime</b>: Returns the time and date this zone computer was last modified (in generalized time format).</li></ul>

---

## Return value

This command returns a field value. The data type depends on the field specified.

## Examples

```
get_zone_computer_field dnsname
```

• • • • •

This example returns the name of the zone computer as listed in DNS:

```
printserver.acme.com
```

## Related commands

Before you use this command, you must have a currently selected zone computer stored in memory. The following commands enable you to view and manage the zone computers:

- `get_zone_computers` returns a Tcl list of the Active Directory names of all zone computers in the current zone.
- `list_zone_computers` lists to `stdout` the zone computers in the current zone.
- `new_zone_computer` creates a new zone computer and stores it in memory.
- `select_zone_computer` retrieves a zone computer from Active Directory and stores it in memory.

After you have a zone computer stored in memory, you can use the following commands to work with that zone computer:

- `delete_zone_computer` deletes the zone computer from Active Directory and from memory.
- `save_zone_computer` saves the zone computer with its current settings to Active Directory.
- `set_zone_computer_field` sets a field value in the currently selected zone computer.

## get\_zone\_computers

Use the `get_zone_computers` command to check Active Directory and return a Tcl list of zone computers defined within the currently selected zone. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use `list_zone_computers` to output the list to `stdout`.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_zone_computers
```

## Abbreviation

gzc

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of zone computers defined in the currently selected zone. Each entry in the list is the security identifier (SID) of a computer that you can use to look up that computer.

## Examples

```
get_zone_computers
```

This example returns the security identifier for each computer:

```
*S-1-5-21-2076040321-3326545908-468068287-1107
```



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and manage the zone computers:

- `list_zone_computers` lists to stdout the zone computers in the current zone.
- `new_zone_computer` creates a new zone computer and stores it in memory.
- `select_zone_computer` retrieves a zone computer from Active Directory and stores it in memory.

After you have a zone computer stored in memory, you can use the following commands to work with that zone computer:

- `delete_zone_computer` deletes the zone computer from Active Directory and from memory.
- `get_zone_computer_field` reads a field value from the currently selected zone computer.
- `save_zone_computer` saves the zone computer with its current settings to Active Directory.
- `set_zone_computer_field` sets a field value in the currently selected zone computer.

## get\_zone\_field

Use the `get_zone_field` command to return the value for a specified field from the currently selected zone stored in memory. The `get_zone_field` command does *not* query Active Directory for this zone. If you change field values using ADEdit without saving the zone to Active Directory, the field value you retrieve using `get_zone_field` won't match the same field value for the zone stored in Active Directory.

## Zone type

Classic and hierarchical



## Syntax

```
get_zone_field field
```

## Abbreviation

gzf

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
field	string	Required. Specifies the case-sensitive name of the field whose value to retrieve.

The data type depends on the field you return. The possible field values are:

- **availableshells:** Returns the shells available to assign to new users in the zone.
- **block.parent.zgroup:** Returns the value of the `block.parent.zgroup` field in the zone object's description.
- **cloudurl:** Returns the name of the cloud instance associated with the selected zone.
- **computers:** Returns the computer group UPN that is assigned to the computer role selected as a zone.
- **createTime:** Returns the time and date this zone was created.
- **customAttr:** Returns the custom text strings that have been set for the zone. This field is only applicable for hierarchical zones.
- **defaultgid:** Returns the default primary group to assign to new users.



- **defaultgecos**: Returns the default GECOS data to assign to new users.
- **defaulthome**: Returns the default home directory to assign to new users.
- **defaultshell**: Returns the default shell to assign to new users.
- **description**: Returns the description of the zone.
- **dn**: Returns the zone's distinguished name.
- **gidnext**: Returns the next GID to use when auto-assigning GID numbers to new groups.
- **gidreserved**: Returns the GID number or range of numbers (1-100) that are reserved.
- **groupname**: Returns the default group name used for new groups in the zone.
- **modifyTime**: Returns the time and date this zone was last modified.
- **nisdomain**: Returns the name of the NIS domain if it has been set.
- **parent**: Returns the distinguished name (DN) of the parent zone for the selected zone.
- **schema**: Returns the schema used in this zone, for example, `std`.
- **sid2iddomainmap**: Returns the domain ID mapping from the selected zone. This field is not supported for auto zones nor classic zones.
- **sfudomain**: Returns the Windows domain name for the SFU zone. Only use this argument if the current zone is a Service for UNIX (sfu) zone.
- **tenantid**: Returns the Centrify Identity Platform tenant ID for the zone. This field is only applicable for hierarchical zones.
- **type**: Returns the type of the zone, for example, `classic4` or `tree`.
- **uidnext**: Returns the next UID to use when auto-assigning UID numbers to new users.
- **uidreserved**: Returns the UID number or range of numbers (1-100) that are reserved.
- **username**: Returns the default user name used for new users in the zone.

For more information about the values returned by these fields, see the Return value section.



## Return value

This command returns the current value for the specified field. The data type depends on the field specified.

This field	Returns
availableshells	Returns the list of shells available to choose from when adding new users to the currently selected zone. The value is a list of shell paths, separated by colons (:). For example, <code>"/bin/bash:/bin/csh:/bin/ksh"</code>
block.parent.zgroup	Returns the value of the <code>block.parent.zgroup</code> field from the zone object's description for the currently selected zone. This field can be <code>true</code> if you want to prevent groups provisioned in the parent zone from being visible in the child zone if they aren't being used. The default value is <code>false</code> .
cloudurl	Returns the fully-qualified URL of the cloud instance associated with the selected zone.
computers	Returns the computer group UPN that is assigned to the computer role if the currently selected zone is a "computer role" zone.
createTime	Returns the time and date this zone was created (in generalized time format).
defaultgid	Returns the default primary group to assign to new users in the currently selected zone. The value can be a specific GID value or include variables.
defaultgecos	Returns the default GECOS data to assign to new users in the currently selected zone. The value can be a string or include variables.
defaulthome	Returns the default home directory to assign to new users in the currently selected zone. The value can be a string that defines the path or include variables.
defaultshell	Returns the default shell to assign to new users in the currently selected zone. The value can be a string that defines the shell or include variables.
description	Returns the description of the zone. If the currently selected zone is a computer role, this field returns the Active Directory description attribute for the <code>msds-AzScope</code> object.
dn	Returns the zone's distinguished name. If the currently selected zone is a computer role, this field returns the Active Directory distinguished name attribute of the <code>msds-AzScope</code> object.
gidnext	Returns the next GID to use when auto-assigning GID numbers to new groups in the currently selected zone.
gidreserved	Returns the GID number or range of numbers (1-100) that are reserved in the currently selected zone.
groupname	Returns the default group name used for new groups in the currently



This field	Returns
	selected zone. You can only return the value for this field if the current zone is a hierarchical zone.
modifyTime	Returns the time and date this zone was last modified (in generalized time format).
nisdomain	Returns the name of the NIS domain if it has been set. The default value is the zone name.
parent	Returns the distinguished name (DN) of the parent zone for the currently selected zone. You can only return the value for this field if the current zone is a hierarchical zone. You can use the option <code>-raw</code> with this field to return the <code>parentLink</code> attribute in the raw <code>Guid@Domain</code> format.
schema	Returns the schema used in this zone, for example, <code>std</code> .
sfudomain	Returns the Windows domain name for the SFU zone. Only use this argument if the current zone is a Service for UNIX (sfu) zone.
sid2iddomainmap	Returns a comma-separated key value pairs string. If an empty string is returned, that means that there's no domain ID mapping for the selected zone.
tenantid	Returns the tenant ID of the cloud instance associated with the selected zone.
type	Returns the type of the currently selected zone. For example, this field returns <code>classic3</code> or <code>classic4</code> for a classic zone or <code>tree</code> for a hierarchical zone.
uidnext	Returns the next UID to use when auto-assigning UID numbers to new users in the currently selected zone.
uidreserved	Returns the UID number or range of numbers (1-100) that are reserved in the currently selected zone.
username	Returns the default user name used for new users in the zone. You can only return the value for this field if the current zone is a hierarchical zone.

## Examples

```
get_zone_field type
```

This example returns the zone type:

```
tree
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the zone:

- `create_zone` creates a new zone in Active Directory.
- `get_zones` returns a Tcl list of all zones within a specified domain.
- `select_zone` retrieves a zone from Active Directory and stores it in memory.

After you have a zone stored in memory, you can use the following commands to work with that zone computer:

- `delegate_zone_right` delegates a zone use right to a specified user or computer.
- `delete_zone` deletes the selected zone from Active Directory and memory.
- `get_child_zones` returns a Tcl list of child zones, computer roles, or computer zones.
- `get_zone_nss_vars` returns the NSS substitution variable for the selected zone.
- `save_zone` saves the selected zone with its current settings to Active Directory.
- `set_zone_field` sets a field value in the currently selected zone.

## `get_zone_group_field`

Use the `get_zone_group_field` command to return the value for a specified field from the currently selected zone group stored in memory. The `get_zone_group_field` command does *not* query Active Directory for the zone group. If you change field values using ADEdit without saving the zone group to Active Directory, the field value you retrieve using `get_zone_group_field` won't match the same field value for the zone group stored in Active Directory.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_zone_group_field field
```

## Abbreviation

gzgf

## Options

This command takes no options.

## Arguments

This command takes the following argument:



---

Argument	Type	Description
field	string	<p>Required. Specifies the case-sensitive name of the field whose value to retrieve. The possible values are:</p> <ul style="list-style-type: none"><li>▪ <b>addn</b>: Returns the distinguished name of the Active Directory group object for the zone group. For example, if the group object is created in the default Users container, this field might return a path similar to <code>CN=pubs-team,CN=Users,DC=ajax,DC=org</code>.</li><li>▪ <b>createTime</b>: Returns the time and date this zone group was created (in generalized time format).</li><li>▪ <b>dn</b>: Returns the distinguished name of the service connection point for the zone group. If the zone is a Services for UNIX (sfu) zone, no value is returned for this field.</li><li>▪ <b>gid</b>: Returns the numeric identifier for the group.</li><li>▪ <b>modifyTime</b>: Returns the time and date this zone group was last modified (in generalized time format).</li><li>▪ <b>name</b>: Returns the group name.</li><li>▪ <b>required</b>: Returns 1 if the zone group is required for members in this zone, or 0 if the group is not required. Users assigned to a required group cannot remove the group from their active set of groups.</li></ul> <p>You can also specify AIX extended attributes as the field to get an extended attribute value for a group. Extended attribute fields start with the <code>aix.</code> prefix. For example, the <code>admin</code> extended attribute can be retrieved by specifying <code>aix.admin</code> as the field.</p>

---

## Return value

This command returns a field value. The data type depends on the field specified.

## Examples

The following example returns the group name.

.....

```
get_zone_group_field name
padmins
```

If the current group is on AIX, you can get AIX group extended attributes and values. For example, to find out if the current group is an administrative group, you can get the `admin` extended attribute:

```
get_zone_group_field aix.admin
true
```

## Related commands

Before you use this command, you must have a currently selected zone group stored in memory. The following commands enable you to view and manage the zone groups:

- [list\\_zone\\_groups](#) lists to `stdout` the zone groups in the current zone.
- [new\\_zone\\_group](#) creates a new zone group and stores it in memory.
- [select\\_zone\\_group](#) retrieves a zone group from Active Directory and stores it in memory.

After you have a zone group stored in memory, you can use the following commands to work with that zone group:

- [delete\\_zone\\_group](#) deletes the selected zone group from Active Directory and from memory.
- [save\\_zone\\_group](#) saves the selected zone group with its current settings to Active Directory.
- [set\\_zone\\_group\\_field](#) sets a field value in the currently selected zone group.

## get\_zone\_groups

Use the `get_zone_groups` command to check Active Directory and return a Tcl list of zone groups defined within the currently selected zone. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use `list_zone_groups` to output the list to `stdout`.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_zone_groups
```

## Abbreviation

```
gzg
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of zone groups defined in the currently selected zone. Each entry in the list is the user principal name (UPN) of a group that you can use to look up that group.

## Examples

```
get_zone_groups
```

This example returns the list of zone groups: `poweradmins@acme.com`  
`auditors@acme.com`



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select zone groups:

- `list_zone_groups` lists to stdout the zone groups in the current zone.
- `new_zone_group` creates a new zone group and stores it in memory.
- `select_zone_group` retrieves a zone group from Active Directory and stores it in memory.

After you have a zone group stored in memory, you can use the following commands to work with that zone group:

- `delete_zone_group` deletes the selected zone group from Active Directory and from memory.
- `get_zone_group_field` reads a field value from the currently selected zone group.
- `save_zone_group` saves the selected zone group with its current settings to Active Directory.
- `set_zone_group_field` sets a field value in the currently selected zone group.

## `get_zone_nss_vars`

Use the `get_zone_nss_vars` command to return a Tcl list containing the NSS substitution variables for the currently selected zone stored in memory. This command only works on hierarchical zones and won't return a value for other zone types.

The `get_zone_nss_vars` command does *not* query Active Directory for this zone. If you change the variables using `set_zone_field` without saving the zone Active Directory, the variable you retrieve using `get_zone_nss_vars` won't match the same field variable for the zone stored in Active Directory.

.....

## Zone type

Hierarchical only

## Syntax

```
get_zone_nss_vars
```

## Abbreviation

```
gznv
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of strings in the form "A=B".

## Examples

```
get_zone_nss_vars
```

This example returns: NSSRANDCOUNT=32000  
NSRANDFILE=/params/nssrand.seed



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone:

- `create_zone` creates a new zone in Active Directory.
- `get_zones` returns a Tcl list of all zones within a specified domain.
- `select_zone` retrieves a zone from Active Directory and stores it in memory.

After you have a zone stored in memory, you can use the following commands to work with that zone:

- `delegate_zone_right` delegates a zone use right to a specified user or computer.
- `delete_zone` deletes the selected zone from Active Directory and memory.
- `get_child_zones` returns a Tcl list of child zones, computer roles, or computer zones.
- `get_zone_field` reads a field value from the currently selected zone.
- `save_zone` saves the selected zone with its current settings to Active Directory.
- `set_zone_field` sets a field value in the currently selected zone.

### `get_zone_user_field`

Use the `get_zone_user_field` command to return the value for a specified field from the currently selected zone user stored in memory. The `get_zone_user_field` command does *not* query Active Directory for the zone user. If you change field values using ADEdit without saving the zone user to Active Directory, the field value you retrieve using `get_zone_user_field` won't match the same field value for the zone user stored in Active Directory.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_zone_user_field field
```

## Abbreviation

gzuf

## Options

This command takes no options.

## Arguments

This command takes the following required argument:

`field` (string type)

Specifies the case-sensitive name of the field whose value to retrieve.

## Argument values

- **addn**: Returns the distinguished name of the Active Directory user object for the zone user. For example, if the user object is created in the default Users container, this field might return a path similar to `CN=amy.adams,CN=Users,DC=ajax,DC=org`.
- **createTime**: Returns the time and date this zone user was created.



- **dn:** Returns the distinguished name of the service connection point for the zone user. If the zone is a Services for UNIX (sfu) zone, no value is returned for this field.
- **enabled:** Returns 1 if the user is enabled, or 0 if the user is disabled. This field is only applicable for users in classic zones. All other zone types use roles.
- **foreign:** If the zone user comes from another forest, this field returns the user principal name of the zone user. Otherwise, this field returns no value.
- **gecos:** Returns information from the GECOS field.
- **gid:** Returns the primary group identifier (GID) for the user.
- **home:** Returns user's home directory.
- **modifyTime:** Returns the time and date this zone user was last modified.
- **shell:** Returns the user's shell type.
- **uid:** Returns the numeric identifier for the user.
- **uname:** Returns the user name.

You can also specify AIX extended attributes as the field to get an extended attribute value for a zone user.

## Return value

This command returns a field value. The data type depends on the field specified.

## Examples

The following example returns the current zone user's user name:

```
get_zone_user_field uname  
adam
```

If the current zone user is on AIX, you can get extended attributes and values. For example:

.....

```
select_zone_user aixul@acme.com  
get_zone_user_field aix.ttys  
u1,u2,u3
```

## Related commands

Before you use this command, you must have a currently selected zone user stored in memory. The following commands enable you to view and select a zone user:

- `get_zone_users` returns a Tcl list of the Active Directory names of all zone users in the current zone.
- `list_zone_users` lists to `stdout` the zone users and their NSS data in the current zone.
- `new_zone_user` creates a new zone user and stores it in memory.
- `select_zone_user` retrieves a zone user from Active Directory and stores it in memory.

After you have a zone user stored in memory, you can use the following commands to work with that zone user:

- `delete_zone_user` deletes the selected zone user from Active Directory and from memory.
- `save_zone_user` saves the selected zone user with its current settings to Active Directory.
- `set_zone_user_field` sets a field value in the currently selected zone user.

## get\_zone\_users

Use the `get_zone_users` command to check Active Directory and return a Tcl list of zone users defined within the currently selected zone. If executed in a script, this command does not output its list to `stdout`, and no output appears in the shell where the script is executed. Use `list_zone_users` to output the list to `stdout`.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_zone_users [-upn]
```

## Abbreviation

gzu

## Options

This command takes the following option:

Option	Description
-upn	Optional. Returns user names in user principal name (UPN) format rather than the default SAMAccount@domain format.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of zone users defined in the currently selected zone. By default, users are listed by SAMAccountName@domain. You can use the -upn option to return users listed by user principal name (UPN). If a zone user is an orphan user—that is, its corresponding Active Directory user no longer exists—the user is listed by its security identifier (SID) instead of the SAMAccountName or user principal name.



## Examples

```
get_zone_users
```

This example returns the list of users: adam.avery brenda.butler  
chris.carter

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone user:

- [list\\_zone\\_users](#) lists to stdout the zone users and their NSS data in the current zone.
- [new\\_zone\\_user](#) creates a new zone user and stores it in memory.
- [select\\_zone\\_user](#) retrieves a zone user from Active Directory and stores it in memory.

After you have a zone user stored in memory, you can use the following commands to work with that zone user:

- [delete\\_zone\\_user](#) deletes the selected zone user from Active Directory and from memory.
- [get\\_zone\\_user\\_field](#) reads a field value from the currently selected zone user.
- [save\\_zone\\_user](#) saves the selected zone user with its current settings to Active Directory.
- [set\\_zone\\_user\\_field](#) sets a field value in the currently selected zone user.

## get\_zones

Use the `get_zones` command to check Active Directory and return a Tcl list of zones within a specified domain. Note that this does not include computer-specific override zones or computer roles.

.....

## Zone type

Classic and hierarchical

## Syntax

```
get_zones domain
```

## Abbreviation

gz

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
domain	string	Required. Specifies the name of the domain for which to return zones.

## Return value

This command returns a Tcl list with the distinguished name for each zone in the specified domain.

## Examples

```
get_zones acme.com
```

This example returns the list of zones in the `acme.com` domain:

• • • • •

```
CN=childzone1,CN=Zones,CN=Centrify,CN=Program  
Data,DC=acme,DC=com  
CN=childzone2,CN=Zones,CN=Centrify,CN=Program  
Data,DC=acme,DC=com  
CN=global,CN=Zones,CN=Centrify,CN=Program  
Data,DC=acme,DC=com
```

## Related commands

The following commands perform actions related to this command:

- [create\\_zone](#) creates a new zone in Active Directory.
- [select\\_zone](#) retrieves a zone from Active Directory and stores it in memory.

After you have a zone stored in memory, you can use the following commands to work with that zone:

- [delegate\\_zone\\_right](#) delegates a zone use right to a specified user or computer.
- [delete\\_zone](#) deletes the selected zone from Active Directory and memory.
- [get\\_child\\_zones](#) returns a Tcl list of child zones, computer roles, or computer zones.
- [get\\_zone\\_field](#) reads a field value from the currently selected zone.
- [get\\_zone\\_nss\\_vars](#) returns the NSS substitution variable for the selected zone.
- [save\\_zone](#) saves the selected zone with its current settings to Active Directory.
- [set\\_zone\\_field](#) sets a field value in the currently selected zone.

## getent\_passwd

Use the `getent_passwd` command to return a Tcl list of local UNIX users that are defined in the `/etc/passwd` file on the AEdit host computer. If the local host is joined to an Active Directory domain, the command also returns

.....

information for the Active Directory users who have a profile in the joined domain and zone.

## Zone type

Not applicable

## Syntax

```
getent_passwd
```

## Abbreviation

gep

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a Tcl list of `/etc/passwd` file entries with all user profile attributes.

## Examples

```
getent_passwd
```

This example returns the contents of the local `/etc/passwd` file:

.....

```
{root x 0 0 root /root /bin/bash} {bin x 1 1 bin /bin
/sbin/nologin}
{daemon x 2 2 daemon /sbin /sbin/nologin}
{adm x 3 4 adm /var/adm /sbin/nologin}
{lp x 4 7 lp /var/spool/lpd /sbin/nologin}
{sync x 5 0 sync /sbin /bin/sync}
{shutdown x 6 0 shutdown /sbin /sbin/shutdown}
```

## Related commands

The following command performs actions related to this command:

- `get_pwnam` searches the `/etc/passwd` file for a UNIX user name and, if found, returns a Tcl list of the profile attributes associated with the user.

## guid\_to\_id

Use the `guid_to_id` command to specify a globally unique identifier (GUID) for a user or group and returns a UID or GID that uses the Apple methodology for automatically generated unique identifiers.

## Zone type

Not applicable

## Syntax

```
guid_to_id guid
```

## Abbreviation

None.



## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
guid	string	Required. Specifies the globally unique identifier for a user or group.

## Return value

This command returns UID or GID for the user or group generated using the Apple mechanism for automatically generating identifiers.

## Examples

```
guid_to_id 763ddbc8-44cc-4a79-83aa-abc899b46aba
```

This example returns the UID for the user associated with the specified globally unique identifier:

```
1983765448
```

## Related commands

The following command performs actions related to this command:

- `principal_to_id` returns a unique UID or GID based on either the Apple methodology or the Centrify Auto Zone methodology for generating numeric identifiers.
- `sid_to_uid` converts a user's security identifier to a numeric identifier (UID).



## help

Use the `help` command to return information about one or more AEdit commands. It's followed by a command pattern that is either the name of a single AEdit command or a string with wild cards that specifies multiple possible commands. The command pattern can also be a command abbreviation.

The command pattern wild cards are:

- `?` for a single character
- `*` for multiple characters

## Zone type

Not applicable

## Syntax

```
help command_pattern
```

## Abbreviation

h

## Options

This command takes no options.

## Arguments

This command takes the following argument:



Argument	Type	Description
command_ pattern	string	Required. Specifies the name of one or more AEdit commands for which to return information. You can specify a command name, command shortcut or use the ? and * wild cards to specify a single character or multiple characters respectively.

## Return value

This command returns information for the specified command or commands. If there's no match for the *command\_pattern* you specify, the command returns nothing.

## Examples

```
help explain_sd
```

This example returns information for the `explain_sd` command.

```
help ?et*
```

This example returns information for the AEdit commands that start with `get` or `set`, such as `get_zones`, `get_zone_field`, `set_zone_field`, and `set_role_field`.

## Related commands

None.

## is\_dz\_enabled

Use this command to check whether authorization is enabled in a currently selected classic zone.

.....

## Zone type

Classic only

## Syntax

```
is_dz_enabled
```

## Abbreviation

```
idze
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns 1 if authorization is enabled in a classic or 0 if authorization is not enabled.

## Examples

```
create_zone classic4 cn=c125,cn=zones,dc=test,dc=net
select_zone cn=c125,cn=zones,dc=test,dc=net
is_dz_enable
0
```

• • • • •

```
manage_dz -on  
is_dz_enable  
1
```

This code example creates a new classic zone, checks that authorization is disabled by default, then enables authorization for the zone.

## Related commands

The following command performs actions related to this command:

- `manage_dz` enables and disables authorization in classic4 zones.

## joined\_get\_user\_membership

Use the `joined_get_user_membership` command to have `adcli` query Active Directory for a list of groups that a specified user belongs to in the domain to which ADEdit's host computer is joined. If the `adcli` query returns groups, this command returns those groups in a Tcl list.

Because this command queries Active Directory through `adcli`, the query might use the `adcli` cache instead of connecting directly to Active Directory. The `adcli` cache isn't guaranteed to be updated with ADEdit activity. Therefore, you might need to execute the Centrify UNIX command `adflush` before using `joined_get_user_membership` to ensure you get the most up-to-date results.

## Zone type

Not applicable

## Syntax

```
joined_get_user_membership user_UPN
```

.....

## Abbreviation

`joined`

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>user_UPN</code>	string	Required. Specifies the user principal name (UPN) of the user to check for group membership.

## Return value

This command returns a Tcl list of groups.

## Examples

```
joined_get_user_membership liz.lemon@acme.com
```

This example returns group membership for `liz.lemon` in the joined domain:

```
acme.com/Users/Domain Users
```

## Related commands

The following commands performs actions related to this command:

- [joined\\_user\\_in\\_group](#) checks Active Directory through `adcli` to see if a user is in a group.

.....

- `get_effective_groups` returns a Tcl list of groups a user belongs to.
- `get_group_members` returns a Tcl list of members in a group.

## joined\_name\_to\_principal

Use the `joined_name_to_principal` command have `adcli` query Active Directory for a UNIX name of a specified user. If the specified user is found, the command returns the associated Active Directory user name in the format of `SAMAccountName@domain`. The command can also optionally return the user principal name (UPN) of the user. This command works only for users within the domain to which ADEdit's host computer is joined through `adcli`.

## Zone type

Not applicable

## Syntax

```
joined_name_to_principal [-upn] UNIX_name
```

## Abbreviation

`jntp`

## Options

This command takes the following option:

Option	Description
<code>-upn</code>	Returns the user's Active Directory name in user principal name (UPN) format.

.....

## Arguments

This command takes the following argument:

Argument	Type	Description
UNIX_name	string	Required. Specifies the UNIX name of a user_ to look for in Active Directory.

## Return value

This command returns the `sAMAccountName@domain` form of the user name if the user is found in Active Directory. If you specify the `-upn` option, this command returns the UPN form of user name.

## Examples

```
joined_name_to_principal -upn adam
```

This example returns the `sAMAccountName@domain` for the UNIX user `adam`:

```
adam.avery@acme.com
```

## Related commands

The following commands performs actions related to this command:

- [principal\\_to\\_dn](#) searches Active Directory for a user principal name (UPN) and, if found, returns the corresponding DN.
- [dn\\_to\\_principal](#) searches Active Directory for a distinguished name and, if found, returns the corresponding UPN.
- [principal\\_from\\_sid](#) searches Active Directory for a security identifier (SID) and returns the security principal associated with the SID.

.....

## joined\_user\_in\_group

Use the `joined_user_in_group` command to have `adcli` query Active Directory to see if a specified user belongs to a specified group. This command works only for users and groups within the domain to which ADEdit's host computer is joined through `adcli`.

Because this command queries Active Directory through `adcli`, the query might use `adcli`'s cache rather than connect directly to Active Directory. The `adcli` cache isn't guaranteed to be updated with ADEdit activity. Therefore, you might need to execute the Centrify UNIX command `adflush` before using `joined_user_in_group` to ensure you get the most up-to-date results.

## Zone type

Not applicable

## Syntax

```
joined_user_in_group user_UPN group_UPN
```

## Abbreviation

`jug`

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
user_UPN	string	Required. Specifies the user principal name (UPN) of the user for which you want to check group membership.
group_UPN	string	Required. Specifies the UPN of the group for which you want to check user membership.

## Return value

This command returns 1 if the user is a member of the group, or 0 if the user is not a member of the group.

## Examples

```
joined_user_in_group martin.moore@acme.com
poweradmins@acme.com
```

This example returns 1 because `martin.moore` is a member of the `poweradmins` group.

## Related commands

The following commands performs actions related to this command:

- `joined_get_user_membership` uses `adcli` to return a Tcl list of groups that a user belongs to.
- `get_effective_groups` checks Active Directory directly and returns a Tcl list of groups a user belongs to.
- `get_group_members` checks Active Directory and returns a Tcl list of members in a group.

## list\_dz\_commands

Use the `list_dz_commands` command to check Active Directory and return a list of UNIX command objects defined within the currently selected zone. If executed in a script, this command outputs its list to `stdout` so that the

• • • • •

output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_dz_commands` to return a Tcl list.

You can only use the `list_dz_commands` command to return UNIX command data for classic4 and hierarchical zones.

## Zone type

Classic and hierarchical

## Syntax

```
list_dz_commands
```

## Abbreviation

```
lsdzc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of UNIX commands defined in the currently selected zone. Each entry in the list contains the following fields, separated by colons (:):



- The name of the UNIX command followed by a slash (/) and the name of the zone where the command is defined.
- The properties of the command.
- Text describing the command.

## Examples

```
list_dz_commands
```

This example returns commands in the following format:

```
root_any/global : * form(0) dzdo_runas(root) flags(16) :  
Run any command as root
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a UNIX command:

- [get\\_dz\\_commands](#) returns a Tcl list of UNIX commands in the current zone.
- [new\\_dz\\_command](#) creates a new UNIX command and stores it in memory.
- [select\\_dz\\_command](#) retrieves a UNIX command from Active Directory and stores it in memory.

After you have a UNIX command stored in memory, you can use the following commands to work with that command:

- [delete\\_dz\\_command](#) deletes the selected command from Active Directory and from memory.
- [get\\_dzc\\_field](#) reads a field value from the currently selected command.
- [save\\_dz\\_command](#) saves the selected command with its current settings to Active Directory.
- [set\\_dzc\\_field](#) sets a field value in the currently selected command.

.....

## list\_local\_groups\_profile

Use the `list_local_groups_profile` command to display a list of local UNIX and Linux group profiles that are defined in the current zone.

### Zone type

Hierarchical only.

### Syntax

```
list_local_groups_profile
```

### Abbreviation

```
lslgp
```

### Options

This command takes no options.

### Arguments

This command takes no arguments.

### Return value

This command returns a list to stdout of the local UNIX and Linux group profiles that are defined in the current zone. Each profile contains the group name, GID, members, and profile flag value.



## Examples

The following example returns a local group profile list.

```
list_local_groups_profile
lam_grp1:3001:lam_usr1:1
lam_grp2:3002:lam_usr2:1
lam_grp3:3003:lam_usr3:3
```

## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- [delete\\_local\\_group\\_profile](#) deletes a local UNIX or Linux group that has a profile defined in the current zone.
- [delete\\_local\\_user\\_profile](#) deletes a local UNIX or Linux user that has a profile defined in the current zone.
- [get\\_local\\_group\\_profile\\_field](#) displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- [get\\_local\\_groups\\_profile](#) displays a TCL list of profiles for local groups that are defined in the current zone.
- [get\\_local\\_user\\_profile\\_field](#) displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- [get\\_local\\_users\\_profile](#) displays a TCL list of profiles for local users that are defined in the current zone.
- [list\\_local\\_users\\_profile](#) displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- [new\\_local\\_group\\_profile](#) creates an object for a local UNIX or Linux group in the currently selected zone.
- [new\\_local\\_user\\_profile](#) creates an object for a local UNIX or Linux user in the currently selected zone.



- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## list\_local\_users\_profile

Use the `list_local_users_profile` command to display a list of local UNIX and Linux user profiles that are defined in the current zone.

### Zone type

Hierarchical only.

### Syntax

```
list_local_users_profile
```

### Abbreviation

```
lslup
```

.....

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of the local UNIX and Linux user profiles that are defined in the current zone. Each profile contains the user name, UID, primary GID, GECOS, home directory, shell, and profile flag value.

## Examples

The following example returns a local user profile list.

```
list_local_users_profile
lam_usr1:2001:2001:lam usr1:/home/lam_usr1:/bin/bash:1
lam_usr2:2002:2002:lam usr2:/home/lam_usr2:/bin/bash:2
lam_usr3:2003:2003:lam usr3:/home/lam_usr3:/bin/bash:3
```

## Related commands

The following related ADEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- [delete\\_local\\_group\\_profile](#) deletes a local UNIX or Linux group that has a profile defined in the current zone.
- [delete\\_local\\_user\\_profile](#) deletes a local UNIX or Linux user that has a profile defined in the current zone.



- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

.....

## list\_nis\_map

Use the `list_nis_map` command to return a list of all map entries within the currently selected NIS map. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_nis_map` to return a Tcl list of NIS map entries.

## Zone type

Not applicable

## Syntax

```
list_nis_map
```

## Abbreviation

lsnm

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of the map entries for the currently selected NIS map. Each map entry in the list contains the following fields

• • • • •

separated by colons (:):

- The key
- The instance number of the key
- The value

## Examples

```
list_nis_map
```

This example returns map entries similar to the following:

```
Finance:1:Hank@acme.com,jane@acme.com,joe@acme.com
```

```
Mktg:1:Mike@acme.com,Sue@acme.com
```

## Related commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and select a NIS map:

- [get\\_nis\\_maps](#) returns a Tcl list of NIS maps in the currently selected zone.
- [list\\_nis\\_maps](#) returns a list to stdout of all NIS maps in the currently selected zone.
- [new\\_nis\\_map](#) creates a new NIS map and stores it in memory.
- [select\\_nis\\_map](#) retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map:

- [add\\_map\\_entry](#) or [add\\_map\\_entry\\_with\\_comment](#) adds a map entry to the currently selected NIS map.
- [delete\\_map\\_entry](#) removes an entry from the currently selected NIS map.



- `delete_nis_map` deletes the selected NIS map from Active Directory and from memory.
- `get_nis_map` or `get_nis_map_with_comment` returns a Tcl list of the map entries in the currently selected NIS map.
- `get_nis_map_field` reads a field value from the currently selected NIS map.
- `list_nis_map_with_comment` lists to `stdout` the map entries in the currently selected NIS map.
- `save_nis_map` saves the selected NIS map with its current entries to Active Directory.

## list\_nis\_map\_with\_comment

Use the `list_nis_map_with_comment` command to return a list of all map entries for the currently selected NIS map and includes the entries' comment. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed.

The command does not return a Tcl list back to the executing script. Use `get_nis_map` or `get_nis_map_with_comment` to return a Tcl list of NIS map entries for parsing or further processing within the script.

## Zone type

Not applicable

## Syntax

```
list_nis_map_with_comment
```

## Abbreviation

```
lsmwc
```

.....

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of the map entries for the currently selected NIS map. Each map entry in the list contains the following fields separated by colons (:):

- The key
- The instance number of the key
- The value
- The comment

## Examples

```
list_nis_map_with_comment
```

This example returns map entries similar to the following:

```
Finance:1:Hank@acme.com,jane@acme.com,joe@acme.com:Finance  
dept staff
```

```
Mktg:1:Mike@acme.com,Sue@acme.com:Marketing dept staff
```

## Related commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and select a NIS map:



- `get_nis_maps` returns a Tcl list of NIS maps in the currently selected zone.
- `list_nis_maps` lists to `stdout` the NIS maps in the currently selected zone.
- `new_nis_map` creates a new NIS map and stores it in memory.
- `select_nis_map` retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map:

- `add_map_entry` or `add_map_entry_with_comment` adds a map entry to the currently selected NIS map.
- `delete_map_entry` removes an entry from the currently selected NIS map.
- `delete_nis_map` deletes the selected NIS map from Active Directory and from memory.
- `get_nis_map` or `get_nis_map_with_comment` returns a Tcl list of the map entries in the currently selected NIS map.
- `get_nis_map_field` reads a field value from the currently selected NIS map.
- `list_nis_map` lists to `stdout` the map entries in the currently selected NIS map.
- `save_nis_map` saves the selected NIS map with its current entries to Active Directory.

## `list_nis_maps`

Use the `list_nis_maps` command to check Active Directory and return a list of NIS maps defined in the currently selected zone. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_nis_maps` to return a Tcl list.

.....

## Zone type

Not applicable

## Syntax

```
list_nis_maps
```

## Abbreviation

```
lsnms
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to stdout of NIS maps defined in the currently selected zone.

## Examples

```
list_nis_maps
```

This example returns the list of NS maps for the zone:

```
Aliases
```

```
Printers
```

## Related commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and select a NIS map:

- `get_nis_maps` returns a Tcl list of NIS maps in the currently selected zone.
- `list_nis_maps` lists to `stdout` the NIS maps in the currently selected zone.
- `new_nis_map` creates a new NIS map and stores it in memory.
- `select_nis_map` retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map:

- `add_map_entry` or `add_map_entry_with_comment` adds a map entry to the currently selected NIS map.
- `delete_map_entry` removes an entry from the currently selected NIS map.
- `delete_nis_map` deletes the selected NIS map from Active Directory and from memory.
- `get_nis_map` or `get_nis_map_with_comment` returns a Tcl list of the map entries in the currently selected NIS map.
- `get_nis_map_field` reads a field value from the currently selected NIS map.
- `list_nis_map` or `list_nis_map_with_comment` lists to `stdout` the map entries in the currently selected NIS map.
- `save_nis_map` saves the selected NIS map with its current entries to Active Directory.

.....

## list\_pam\_apps

Use the `list_pam_apps` command to check Active Directory and return a list of PAM application rights defined in the currently selected zone. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_pam_apps` to return a Tcl list.

You can only use the `list_pam_apps` command to return PAM application rights for classic4 and hierarchical zones.

## Zone type

Classic and hierarchical

## Syntax

```
list_pam_apps
```

## Abbreviation

```
lspa
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

.....

## Return value

This command returns a list to `stdout` of PAM application rights defined in the currently selected zone. Each entry contains the following fields, separated by colons (:):

- The name of the PAM access right followed by a slash (/) and the zone in which the PAM access right is defined.
- The name of one or more PAM applications to which the right applies.
- Text describing the PAM application object.

## Examples

```
list_pam_apps
```

This example returns a list of PAM application access rights for the selected zone (the following is a subset of the default predefined rights):

```
dzssh-all/global : dzssh-* : All of ssh services
dzssh-exec/global : dzssh-exec : Command execution
dzssh-scp/global : dzssh-scp : scp
dzssh-sftp/global : dzssh-sftp : sftp
dzssh-shell/global : dzssh-shell : Terminal tty/pty
dzssh-tunnel/global : dzssh-tunnel : Tunnel device
forwarding
dzssh-X11-forwarding/global : dzssh-x11-forwarding : X11
forwarding
login-all/global : * : Predefined global PAM permission. Do
not delete.
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a PAM application object:

- `get_pam_apps` returns a Tcl list of PAM applications in the current zone.
- `new_pam_app` creates a new PAM application and stores it in memory.



- `select_pam_app` retrieves a PAM application from Active Directory and stores it in memory.

After you have a PAM application object stored in memory, you can use the following commands to work with that PAM application:

- `delete_pam_app` deletes the selected PAM application from Active Directory and from memory.
- `get_pam_field` reads a field value from the currently selected PAM application.
- `save_pam_app` saves the selected PAM application with its current settings to Active Directory.
- `set_pam_field` sets a field value in the currently selected PAM application.

## list\_pending\_zone\_groups

Use the `list_pending_zone_groups` command to check Active Directory and return a list of pending import groups for the currently selected zone. Pending import groups are group profiles that have been imported from Linux or UNIX computers, but not yet mapped to any Active Directory group. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_pending_zone_groups` to return a Tcl list.

## Zone type

Classic and hierarchical

## Syntax

```
list_pending_zone_groups
```

## Abbreviation

lpzg

.....

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of pending import groups for the currently selected zone. Each entry in the list contains the following fields, separated by colons (:):

- Distinguished name (DN) of the pending import group as it is stored in Active Directory. The distinguished name for each pending import group includes a prefix that consists of "PendingGroup" and the globally unique identifier (GUID) for the group.
- UNIX group name.
- Numeric group identifier (GID).

## Examples

```
list_pending_zone_groups
```

This example returns the list of groups similar to this:

```
CN=PendingGroup_573135e7-edd9-46b9-9cbd-  
c839570a90c8,CN=Groups, CN=bean_  
pz,CN=Zones,CN=Centrify,DC=win2k3,DC=test:root:0  
CN=PendingGroup_7878065a-4d2f-4749-8f3b-  
6ffe24303f6a,CN=Groups, CN=bean_  
pz,CN=Zones,CN=Centrify,DC=win2k3,DC=test:unixgrp:5000
```



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following command performs actions related to this command:

- `get_pending_zone_groups` returns a Tcl list of the pending import groups in the current zone.

## `list_pending_zone_users`

Use the `list_pending_zone_users` command to check Active Directory and return a list of pending import users for the currently selected zone. Pending import users are user profiles that have been imported from Linux or UNIX computers, but not yet mapped to any Active Directory user. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_pending_zone_users` to return a Tcl list.

## Zone type

Classic and hierarchical

## Syntax

```
list_pending_zone_users
```

## Abbreviation

lpzu

## Options

This command takes no options.

.....

## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of pending import users for the currently selected zone. Each entry in the list contains the following fields, separated by colons (:):

- Distinguished name (DN) of the pending import user as it is stored in Active Directory. The distinguished name for each pending import user includes a prefix that consists of "PendingUser" and the globally unique identifier (GUID) for the user.
- UNIX user name.
- Numeric user identifier (UID).
- Numeric primary group identifier (GID).
- Personal information from the GECOS field.
- Home directory.
- Default login shell.

## Examples

```
list_pending_zone_users
```

This example returns the list of groups similar to this:

```
CN=PendingUser_09024f3a-6abc-4666-a127-722f9fe0e0bf,CN=Users,CN=finance,  
  
CN=Zones,CN=Centrify,DC=win2k3,DC=test:root:0:0:root:/root:/bin/bash:  
CN=PendingUser_0b9fe038-1325-438f-8529-cb190ab5914a,CN=Users,CN=finance,  
CN=Zones,CN=Centrify,DC=win2k3,DC=test:bean:6001:5000:bean.zhang:/home/bean:/bin/bash:
```



## Related commands

The following command performs actions related to this command:

- `get_pending_zone_users` returns a Tcl list of the pending import users in the current zone.

## list\_role\_assignments

Use the `list_role_assignments` command to check Active Directory and return a list of role assignments defined within the currently selected zone. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_role_assignments` to return a Tcl list.

If you do not specify an option, the command returns the current users and groups in the zone with a role assignment using the default `SAMAccount@domain` format.

You can only use the `list_role_assignments` command to return role assignments for classic4 and hierarchical zones.

## Zone type

Classic and hierarchical

## Syntax

```
list_role_assignments [-upn] [-visible] [-user] [-group] [-invalid]
```

## Abbreviation

```
lsra
```



## Options

This command takes the following options:

Option	Description
-upn	Optional. Returns user names in user principal name (UPN) format rather than the default <code>SAMAccount@domain</code> format.
-visible	Returns a list to <code>stdout</code> of the visible role assignments in the zone. Use this option if you only want to return role assignments for the roles that are identified as visible. This option is only applicable in hierarchical zones.
-user	Returns a list to <code>stdout</code> of the current users in the zone with a role assignment. Use this option if you only want to return valid users with a role assignment.
-group	Returns a list to <code>stdout</code> of the current groups in the zone with a role assignment. Use this option if you only want to return valid groups with a role assignment.
-invalid	Returns a list to <code>stdout</code> of any invalid role assignments in the zone. A role assignment is invalid if it specifies a group or user that no longer exists. Use this option if you only want to return invalid role assignments.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of role assignments defined in the currently selected zone. Each entry in the list provides the following information:

- The user or group to whom the role assignment applies by `SAMAccount@domain` name or user principal name.
- The name of the role assigned followed by a slash (/) and the zone where the role is defined.

## Examples

```
>bind pistols.org
```

.....

```
>select_zone  
"cn=northamerica,cn=zones,ou=centrify,dc=pistolas,dc=org"  
  
>list_role_assignments
```

This example returns the role assignments for the `northamerica` zone:

```
Domain Users@pistolas.org: Window Login/northamerica  
adm-sf@pistolas.org: UNIX Login/northamerica  
rey@pistolas.org: UNIX Login/northamerica  
maya@pistolas.org: SQLAdmin/northamerica
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a role assignment:

- `get_role_assignments` returns a Tcl list of role assignments in the current zone.
- `new_role_assignment` creates a new role assignment and stores it in memory.
- `select_role_assignment` retrieves a role assignment from Active Directory and stores it in memory.

After you have a role assignment stored in memory, you can use the following commands to work with that role assignment:

- `delete_role_assignment` deletes the selected role assignment from Active Directory and from memory.
- `get_role_assignment_field` reads a field value from the currently selected role assignment.
- `save_role_assignment` saves the selected role assignment with its current settings to Active Directory.
- `set_role_assignment_field` sets a field value in the currently selected role assignment.



## list\_role\_rights

Use the `list_role_rights` command to return a list of all UNIX commands and PAM application rights set within the currently selected role. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script.

The `list_role_rights` command does *not* query Active Directory for the role. If you change commands or PAM applications using ADEdit without saving the role to Active Directory, commands and PAM applications you retrieve using `list_role_rights` won't match those stored in Active Directory.

You can only use `list_role_rights` to return role rights for classic4 and hierarchical zones.

## Zone type

Classic and hierarchical

## Syntax

```
list_role_rights
```

## Abbreviation

```
lsrr
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.



## Return value

This command returns a list to `stdout` of the PAM application and UNIX command rights that are defined for the currently selected role.

Each entry lists the name of the application or command right, the attributes of the application or command, and any descriptive text.

## Examples

```
list_role_rights
```

This example returns the list of PAM application and UNIX command rights:

```
dzssh-all/northamerica : dzssh-exec : Command execution
login-all/seattle : * : Predefined global PAM permission.
Do not delete.
cron-exec/seattle : cron form(0) dzdo_runas(admin) flags
(16) ;
```

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select a role:

- [get\\_roles](#) returns a Tcl list of roles in the current zone.
- [list\\_roles](#) returns a list of all roles in the currently selected zone.
- [new\\_role](#) creates a new role and stores it in memory.
- [select\\_role](#) retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with that role:

- [add\\_command\\_to\\_role](#) adds a UNIX command right to the current role.
- [add\\_pamapp\\_to\\_role](#) adds a PAM application right to the current role.
- [delete\\_role](#) deletes the selected role from Active Directory and from memory.



- `get_role_apps` returns a Tcl list of the PAM application rights associated with the current role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the current role.
- `get_role_field` reads a field value from the current role.
- `remove_command_from_role` removes a UNIX command from the current role.
- `remove_pamapp_from_role` removes a PAM application from the current role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the current role.

## list\_roles

Use the `list_roles` command to check Active Directory and return a list of roles defined in the currently selected zone. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_roles` to return a Tcl list.

You can only use `list_roles` to return role information for classic4 and hierarchical zones.

## Zone type

Classic and hierarchical

## Syntax

```
list_roles
```

.....

## Abbreviation

`lsr`

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to stdout of roles defined in the currently selected zone.

## Examples

```
list_roles
```

This example returns the list of roles for the zone:

```
Rescue - always permit login  
listed  
scp  
sftp  
UNIX Login  
Windows Login  
winscp
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a



role:

- `get_roles` returns a Tcl list of roles in the current zone.
- `new_role` creates a new role and stores it in memory as the currently selected role.
- `select_role` retrieves a role from Active Directory and stores it in memory as the selected role.

After you have a role stored in memory, you can use the following commands to work with that role:

- `add_command_to_role` adds a UNIX command right to the current role.
- `add_pamapp_to_role` adds a PAM application right to the current role.
- `delete_role` deletes the selected role from Active Directory and from memory.
- `get_role_apps` returns a Tcl list of the PAM application rights associated with the current role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the current role.
- `get_role_field` reads a field value from the current role.
- `list_role_rights` returns a list of all UNIX command and PAM application rights associated with the current role.
- `remove_command_from_role` removes a UNIX command from the current role.
- `remove_pamapp_from_role` removes a PAM application from the current role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the current role.

## `list_rs_commands`

Use the `list_rs_commands` command to print a list of the restricted shell commands that are defined for the currently selected zone. This command retrieves information from Active Directory and to returns the list of

.....

restricted shell commands to stdout. If you want to return a Tcl list of restricted shell commands, use [get\\_rs\\_commands](#).

## Zone type

Classic only

## Syntax

```
list_rs_commands
```

## Abbreviation

```
lsrsc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a list of restricted shell commands for the currently selected zone.

## Examples

```
list_rs_commands
```

.....

This command returns the list of restricted shell commands and attributes similar to this:

```
rseid1/c123 : id form(0) dzsh_runas($) umask(77) path
(USERPATH) flags(0) :
rseid2/c123 : id2 form(0) dzsh_runas($) pri(1) umask(77)
path(USERPATH) flags(0) : id2
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the restricted shell command to work with:

- [get\\_rs\\_commands](#) returns a Tcl list of restricted shell commands in the current zone.
- [new\\_rs\\_command](#) creates a new restricted shell command and stores it in memory.
- [select\\_rs\\_command](#) retrieves a restricted shell command from Active Directory and stores it in memory.

After you have a restricted shell command stored in memory, you can use the following commands to work with that restricted shell:

- [delete\\_rs\\_command](#) deletes the selected command from Active Directory and from memory.
- [get\\_rsc\\_field](#) reads a field value from the currently selected command.
- [save\\_rs\\_command](#) saves the selected command with its current settings to Active Directory.
- [set\\_rsc\\_field](#) sets a field value in the currently selected command.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and manage the restricted shell commands:



- `delete_rs_command` deletes the selected command from Active Directory and from memory.
- `new_rs_command` creates a new restricted shell command and stores it in memory.
- `save_rs_command` saves the selected restricted shell command with its current settings to Active Directory.
- `select_rs_command` retrieves a restricted shell command from Active Directory and stores it in memory.

After you have a restricted shell command stored in memory, you can use the following commands to work with its fields:

- `get_rsc_field` reads a field value from the current restricted shell command.
- `set_rsc_field` sets a field value in the current restricted shell command.

## `list_rs_envs`

Use the `list_rs_envs` command to check Active Directory and print a list of restricted shell environments defined within the currently selected zone to stdout. Use the `get_rs_envs` command to return a Tcl list.

## Zone type

Classic only

## Syntax

```
list_rs_envs
```

## Abbreviation

```
lsrse
```

.....

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command prints the list of restricted shell environments to `stdout`. It has no return value.

## Examples

```
list_rs_envs
```

This example displays the list of restricted shell environments.

```
restrict_env1
```

```
restrict_env2
```

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with restricted shell environments:

- `get_rs_envs` returns a Tcl list of restricted shell environments.
- `new_rs_env` creates a new restricted shell environment and stores it in memory.
- `select_rs_env` retrieves a restricted shell environment from Active Directory and stores it in memory.



After you have a restricted shell environment stored in memory, you can use the following commands to work with its fields:

- `delete_rs_env` deletes the current restricted shell environment from Active Directory and from memory.
- `get_rse_field` reads a field value from the current restricted shell environment.
- `save_rs_env` saves the restricted shell environment to Active Directory.
- `set_rse_field` sets a field value in the current restricted shell environment.

## list\_zone\_computers

Use the `list_zone_computers` command to check Active Directory and return a list of zone computers defined within the currently selected zone. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_zone_computers` to return a Tcl list.

## Zone type

Classic and hierarchical

## Syntax

```
list_zone_computers
```

## Abbreviation

```
lszc
```

## Options

This command takes no options.



## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of zone computers defined in the currently selected zone. Each zone computer entry includes the following fields, separated by colons (:):

- User principal name (UPN) of the computer.
- Number of CPUs in the computer and the version of Centrify software installed on the computer.
- Name of the computer in DNS.

## Examples

```
list_zone_computers
```

This example returns the list of computers similar to this:

```
printserv$@acme.com:cpus (1) agentVersion (CentrifyDC  
5.0.0): printserv.acme.com
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone computer:

- [get\\_zone\\_computers](#) returns a Tcl list of the Active Directory names of all zone computers in the current zone.
- [new\\_zone\\_computer](#) creates a new zone computer and stores it in memory.
- [select\\_zone\\_computer](#) retrieves a zone computer from Active Directory and stores it in memory.



After you have a zone computer stored in memory, you can use the following commands to work with that zone computer:

- `delete_zone_computer` deletes the zone computer from Active Directory and from memory.
- `get_zone_computer_field` reads a field value from the currently selected zone computer.
- `save_zone_computer` saves the zone computer with its current settings to Active Directory.
- `set_zone_computer_field` sets a field value in the currently selected zone computer.

## list\_zone\_groups

Use the `list_zone_groups` command to check Active Directory and return a list of zone groups defined in the currently selected zone. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_zone_groups` to return a Tcl list.

## Zone type

Classic and hierarchical

## Syntax

```
list_zone_groups
```

## Abbreviation

```
lszg
```

.....

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of zone groups defined in the currently selected zone. Each entry in the list contains the following fields, separated by colons (:):

- User principal name of the zone group as it is stored in Active Directory.
- UNIX group name.
- Numeric group identifier (GID).
- The string "Required" if the "Users are required to be members of this group" option is set for the group.

## Examples

```
list_zone_groups
```

This example returns the list of groups similar to this:

```
sf-admins@pistolas-org:sfadmins:10F24  
sf-apps@pistolas.org:sf-apps:2201
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select zone groups:



- `get_zone_groups` returns a Tcl list of the Active Directory names of the zone groups in the current zone.
- `new_zone_group` creates a new zone group and stores it in memory.
- `select_zone_group` retrieves a zone group from Active Directory and stores it in memory.

After you have a zone group stored in memory, you can use the following commands to work with that zone group:

- `delete_zone_group` deletes the selected zone group from Active Directory and from memory.
- `get_zone_group_field` reads a field value from the currently selected zone group.
- `save_zone_group` saves the selected zone group with its current settings to Active Directory.
- `set_zone_group_field` sets a field value in the currently selected zone group.

## list\_zone\_users

Use the `list_zone_users` command to check Active Directory and return a list of zone users defined in the currently selected zone. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use `get_zone_users` to return a Tcl list.

## Zone type

Classic and hierarchical

## Syntax

```
list_zone_users [-upn]
```



## Abbreviation

lszu

## Options

This command takes the following option:

Option	Description
-upn	Optional. Returns user names in user principal name (UPN) format rather than the default sAMAccount@domain format.

## Arguments

This command takes no arguments.

## Return value

This command returns a list to `stdout` of zone users for the currently selected zone. Each entry in the list contains the following user profile fields separated by colons (:):

- sAMAccountName@domain or the UPN of the zone user as it is stored in Active Directory.  
If the Active Directory user no longer exists for a zone user, the command returns the security identifier (SID) of the orphan user.
- UNIX user name.
- Numeric user identifier (UID).
- Numeric identifier for the user's primary group (GID).  
If the GID has the number 2147483648 (which is 80000000 hex) it means that the UID is being used as the GID. (This can occur in hierarchical zones.)
- Personal information from the GECOS field.

• • • • •

- The user's home directory.
- The user's default login shell.
- Whether the user is enabled or disabled (in classic zones only).

## Examples

```
list_zone_users
```

This example returns the list of users similar to this:

```
adam.avery@acme.com:adam:10001:10001:%{u:samaccountname}:%  
{home}/%{user}:%{shell}:  
ben.brown@acme.com:brenda:10002:10002:%{u:samaccountname}:%  
{home}/%{user}:%{shell}:  
chris.cain@acme.com:chris:10003:10003:%{u:samaccountname}:%  
{home}/%{user}:%{shell}:
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select zone users:

- [get\\_zone\\_users](#) returns a Tcl list of the Active Directory names of zone users in the current zone.
- [new\\_zone\\_user](#) creates a new zone user and stores it in memory.
- [select\\_zone\\_user](#) retrieves a zone user from Active Directory and stores it in memory.

After you have a zone user stored in memory, you can use the following commands to work with that zone user:

- [delete\\_zone\\_user](#) deletes the selected zone user from Active Directory and from memory.
- [get\\_zone\\_user\\_field](#) reads a field value from the currently selected zone user.
- [save\\_zone\\_user](#) saves the selected zone user with its current settings to



Active Directory.

- `set_zone_user_field` sets a field value in the currently selected zone user.

## manage\_dz

Use the `manage_dz` command to enable or disable authorization in classic zones. In classic zones, authorization-related features are disabled by default, and the authorization store that is required for managing rights, roles, and restricted environment is not available in Active Directory.

To enable authorization in classic zones using ADEdit, you can run the `manage_dz -on` command. This command creates the authorization store if it does not exist, and sets the zone property that enables privilege elevation service features.

To disable authorization in a classic zone, you can run the `manage_dz -off` command. Running this command disables authorization services. The command does not remove any existing authorization data from Active Directory.

## Zone type

Classic only

## Syntax

```
manage_dz [-on|-off]
```

## Abbreviation

mnz

## Options

This command takes the following options:



Option	Description
-on	Enables authorization for the currently selected zone and creates the authorization data store if it not currently defined in Active Directory.
-off	Disables authorization for the currently selected zone. This option does not remove any data from the authorization data store if it currently exists.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_zone classic4 cn=c125,cn=zones,dc=ross,dc=net
select_zone cn=c125,cn=zones,dc=ross,dc=net
is_dz_enable
0
manage_dz -on
is_dz_enable
1
```

This code example creates a zone, checks that authorization is disabled by default, then enables authorization for the zone.

## Related commands

The following command performs actions related to this command:

- [is\\_dz\\_enabled](#) checks whether authorization is currently enabled for a zone.

.....

## move\_object

Use the `move_object` command to move the selected object to the specified location. The new location must be in the same domain. You cannot use this command to move an object to another domain. You do not need to save the object after moving it.

### Zone type

Not applicable

### Syntax

```
move_object destination
```

### Abbreviation

mvo

### Options

This command takes no options.

### Arguments

This command takes the following argument:

Argument	Type	Description
<code>destination</code>	string	Required. Specifies the distinguished name of the new location.

### Return value

This command returns nothing if it runs successfully.

.....

## Example

The following commands move the ApacheAdmins group from the Groups container in the Global zone to the Groups container in the US zone.

```
select_object  
"cn=ApacheAdmins@demo.test,cn=Groups,cn=Global,cn=Zones,cn=  
Centrify,dc=demo,dc=test"  
mvo "cn=Groups,cn=US,cn=Zones,ou=Centrify,dc=demo,dc=test"
```

## Related commands

The following command performs actions related to this command:

- `select_object` selects the object you want to move.

## new\_dz\_command

Use the `new_dz_command` command to create a new UNIX command object for the current zone and sets the new command as the currently selected command in memory. The new command has no field values set. The `new_dz_command` does *not* save the new command to Active Directory. To save the UNIX command, you must first set at least the "command" field using `set_dzc_field`, then use `save_dz_command`. If you don't save a new UNIX command, it will disappear when you select a new command or when the ADEdit session ends.

You can only use the `new_dz_command` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

.....

## Syntax

```
new_dz_command name
```

## Abbreviation

```
newdzc
```

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
name	string	Required. Specifies the name to assign to the new UNIX command.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
new_dz_command account_manager
```

This example creates a new UNIX command named `account_manager` in the current zone.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select UNIX commands:

- `get_dz_commands` returns a Tcl list of UNIX commands in the current zone.
- `list_dz_commands` returns a list of all UNIX commands in the currently selected zone.
- `select_dz_command` retrieves a UNIX command from Active Directory and stores it in memory.

After you have a UNIX command stored in memory, you can use the following commands to work with that command:

- `delete_dz_command` deletes the selected command from Active Directory and from memory.
- `get_dzc_field` reads a field value from the currently selected command.
- `save_dz_command` saves the selected command with its current settings to Active Directory.
- `set_dzc_field` sets a field value in the currently selected command.

## new\_local\_group\_profile

Use the `new_local_group_profile` command to create an object for a local UNIX or Linux group in the currently selected zone. After you create the group object, it is automatically selected for editing with the `set_local_group_profile_field` command. That is, you do not need to execute the `select_local_group_profile` command to select the new group prior to defining profile fields. After you create the new group, save it by executing the `save_local_group_profile` command.

When the group profile is complete and the `profileflag` field is set to 1 (enabled), the profile is added to `/etc/group` on each UNIX and Linux computer in the zone at the next local account refresh interval. A group profile must have the following fields (attributes) to be considered complete:



- A unique numeric identifier (GID).
- A group name.

See the *Administrator's Guide for Linux and UNIX* for more details about creating local group profiles.

## Zone type

Hierarchical only.

## Syntax

```
new_local_group_profile group_name
```

## Abbreviation

newlgp

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
group_name	string	Required. Specifies the UNIX name of the new local group to create in the zone.

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example shows a typical sequence of commands to create an object for the local UNIX or Linux group `marketing` in the currently selected zone. This command sequence creates a complete group profile, and sets the profile flag to 1 (enabled) so that the profile is added to `/etc/group` at the next local account update interval.

```
new_local_group_profile marketing
set_local_group_profile_field gid 3004
set_local_group_profile_field member lam_usr4
set_local_group_profile_field profileflag 1
save_local_group_profile
```

## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- `delete_local_group_profile` deletes a local UNIX or Linux group that has a profile defined in the current zone.
- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.



- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## `new_local_user_profile`

Use the `new_local_user_profile` command to create an object for a local UNIX or Linux user in the currently selected zone. After you create the user object, it is automatically selected for editing with the `set_local_user_profile_field` command. That is, you do not need to execute the `select_local_user_profile` command to select the new user prior to defining profile fields. After you create the new user, save it by executing the `save_local_user_profile` command.

**Note** Unlike local groups, which are visible by default, you must explicitly assign local users to a visible role. If you do not assign a local user to a visible role, the user profile defined in the zone object is not updated in `/etc/passwd` on local computers. A predefined visible role for local users, `local_listed`, is provided to make local users visible. After you



**Note** create a local user profile, you must assign the local user to the local listed role, or to another visible role. You can use the `select_role_assignment` and `new_role_assignment` ADEdit commands to make role assignments.

When the user profile is complete and the `profileflag` field is set to 1 (enabled) or 2 (disabled), the profile is added to `/etc/passwd` on each UNIX and Linux computer in the zone at the next local account refresh interval.

A user profile must have the following fields (attributes) to be considered complete:

- A user name (the UNIX login name).
- A unique numeric user identifier (UID).
- The user's primary group profile numeric identifier (GID).
- The default home directory for the user.
- The default login shell for the user.

Note that the GECOS field is not required.

See the *Administrator's Guide for Linux and UNIX* for more details about creating local user profiles.

## Zone type

Hierarchical only.

## Syntax

```
new_local_user_profile user_name
```

## Abbreviation

newlup

.....

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
user_name	string	Required. Specifies the UNIX name of the new local user to create in the zone.

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example shows a typical sequence of commands to create an object for the local UNIX or Linux user `lam_usr4` in the currently selected zone. This command sequence creates a complete user profile, sets the profile flag to 1 (enabled), and adds the user to the `local listed` role so that the profile is added to `/etc/passwd` at the next local account update interval.

```
new_local_user_profile lam_usr4
set_local_user_profile_field uid 2004
set_local_user_profile_field gid 2004
set_local_user_profile_field shell /bin/bash
set_local_user_profile_field home /home/lam_usr4
set_local_user_profile_field profileflag 1
save_local_user_profile
select_role_assignment local listed
new_role_assignment lam_usr4
```



## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- `delete_local_group_profile` deletes a local UNIX or Linux group that has a profile defined in the current zone.
- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.



- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## new\_nis\_map

Use the `new_nis_map` command to create a new NIS map for the current zone and set the new NIS map as the currently selected NIS map in memory. The new NIS map has no map entries.

The `new_nis_map` does *not* save the new NIS map to Active Directory. To save the new map, you must use `save_nis_map`. If you don't save a new NIS map, it will disappear when you select a new NIS map or when the ADEdit session ends.

## Zone type

Not applicable

## Syntax

```
new_nis_map [-automount] map
```

## Abbreviation

newnm



## Options

This command takes the following option:

Option	Description
- automount	Specifies that the new NIS map is an <b>automount</b> map.  For most NIS maps, the map name defines the type of map you are creating. For example, if you create a new NIS map with the name <b>netgroup</b> , it must be a NIS <b>netgroup</b> map and contain valid <b>netgroup</b> entries. However, you can specify any name for NIS <b>automount</b> maps. Use this option to identify <b>automount</b> maps that have a name other than <b>automount</b> .

## Arguments

This command takes the following argument:

Argument	Type	Description
map	string	Required. Specifies the name of the new NIS map.  For most NIS maps, the map name defines the type of map you are creating. For example, if you create a new NIS map with the name <b>netgroup</b> , it must be a NIS <b>netgroup</b> map and contain valid <b>netgroup</b> entries. For information about the type of NIS maps you can create, see the <i>Network Information Service Administrator's Guide</i> .

## Return value

This command returns nothing if it runs successfully.

## Examples

The following command creates the NIS map "Printers" in the current zone.

```
new_nis_map Printers
```



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select NIS maps:

- `get_nis_maps` returns a Tcl list of NIS maps in the current zone.
- `list_nis_maps` lists to `stdout` the NIS maps in the current zone.
- `select_nis_map` retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map:

- `add_map_entry` or `add_map_entry_with_comment` adds an entry to the current NIS map stored in memory.
- `delete_map_entry` removes an entry from the current NIS map.
- `delete_nis_map` deletes the selected NIS map from Active Directory and from memory.
- `get_nis_map` or `get_nis_map_with_comment` returns a Tcl list of the map entries in the current NIS map.
- `get_nis_map_field` reads a field value from the current NIS map.
- `list_nis_map` or `list_nis_map_with_comment` lists to `stdout` the map entries in the current NIS map.
- `save_nis_map` saves the selected NIS map with its current entries to Active Directory.

## new\_object

Use the `new_object` command to create a new Active Directory object and set the new object as the currently selected Active Directory object in memory. The new object has no field values set. The `new_object` command does *not* save the new object to Active Directory. To save the new object, you must use `save_object`. If you don't save a new object, it will disappear when you select a new object or when the ADEdit session ends.

.....

The `new_object` command does not check to see if the new object conforms to Active Directory's expectations for the new object in the location you specify. Active Directory will report any errors when you try to save the object.

## Zone type

Not applicable

## Syntax

```
new_object dn
```

## Abbreviation

newo

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
dn	DN	Required. Specifies the distinguished name for the new object.

## Return value

This command returns nothing if it runs successfully.



## Examples

```
new_object "ou=Centrify,cn=Program Data,dc=acme,dc=com"
```

This example creates a new organizational unit `centrify` in the container `Program Data` in the domain `acme.com` and stores it in memory as the currently selected Active Directory object.

## Related commands

The following commands enable you to view and select Active Directory objects:

- [get\\_objects](#) performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of objects matching the specified search criteria.
- [select\\_object](#) retrieves an object with its attributes from Active Directory and stores it in memory.

After you have an object stored in memory, you can use the following commands to work with that object:

- [add\\_object\\_value](#) adds a value to a multi-valued field attribute of the currently selected Active Directory object.
- [delete\\_object](#) deletes the selected Active Directory object from Active Directory and from memory.
- [delete\\_sub\\_tree](#) deletes an Active Directory object and all of its children from Active Directory.
- [get\\_object\\_field](#) reads a field value from the currently selected Active Directory object.
- [remove\\_object\\_value](#) removes a value from a multi-valued field attribute of the currently selected Active Directory object.
- [save\\_object](#) saves the selected Active Directory object with its current settings to Active Directory.
- [set\\_object\\_field](#) sets a field value in the currently selected Active Directory object.



## new\_pam\_app

Use the `new_pam_app` command to create a new PAM application right for the current zone and set the new PAM application as the currently selected PAM application in memory. The new PAM application has no field values set.

The `new_pam_app` does *not* save the new PAM application to Active Directory. To save the PAM application right, you must first set at least the “application” field using `set_pam_field`, then use `save_pam_app`. If you don’t save a new PAM application, it will disappear when you select a new PAM application or when the ADEdit session ends.

You can only use the `new_pam_app` to create PAM application rights if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
new_pam_app name
```

## Abbreviation

newpam

## Options

This command takes no options.

## Arguments

This command takes the following argument:



---

Argument	Type	Description
name	string	Required. Specifies the name to assign to the new PAM application access right.

---

## Return value

This command returns nothing if it runs successfully.

## Examples

```
new_pam_app basic
```

This example creates a new PAM application access right named `basic` in the current zone.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select PAM application rights:

- `get_pam_apps` returns a Tcl list of PAM application rights in the current zone.
- `list_pam_apps` lists to stdout the PAM application rights in the currently selected zone.
- `select_pam_app` retrieves a PAM application right from Active Directory and stores it in memory.

After you have a PAM application right stored in memory, you can use the following commands to work with that PAM application right:

- `delete_pam_app` deletes the selected PAM application right from Active Directory and from memory.
- `get_pam_field` reads a field value from the currently selected PAM application right.



- `save_pam_app` saves the selected PAM application right with its current settings to Active Directory.
- `set_pam_field` sets a field value in the currently selected PAM application right.

## new\_role

Use the `new_role` command to create a new role for the current zone and set the new role as the currently selected role in memory. The new role has no field values set. The `new_role` command does *not* save the new role to Active Directory. To save the new role, you must use `save_role`. If you don't save a new role, it will disappear when you select another role or when the ADEdit session ends.

You can only use the `new_role` to create a role if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
new_role name
```

## Abbreviation

```
newr
```

## Options

This command takes no options.



## Arguments

This command takes the following argument:

Argument	Type	Description
name	string	Required. Specifies the name to assign to the new role.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
new_role customerservice
```

This example creates a new role named `customerservice` in the current zone.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select roles:

- `get_roles` returns a Tcl list of roles in the current zone.
- `list_roles` lists to `stdout` the roles in the current zone.
- `select_role` retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with that role:

- `add_command_to_role` adds a UNIX command to the current role.
- `add_pamapp_to_role` adds a PAM application to the current role.
- `delete_role` deletes the selected role from Active Directory and from memory.



- `get_role_apps` returns a Tcl list of the PAM applications associated with the currently selected role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the current role.
- `get_role_field` reads a field value from the currently selected role.
- `list_role_rights` returns a list of all UNIX commands and PAM application rights associated with the current role.
- `remove_command_from_role` removes a UNIX command from the current role.
- `remove_pamapp_from_role` removes a PAM application from the current role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the currently selected role.

## new\_role\_assignment

Use the `new_role_assignment` command to create a new role assignment for the current zone and set the new role assignment as the currently selected role assignment in memory. The new role assignment has no field values set.

The `new_role_assignment` command does *not* save the new role assignment to Active Directory. To save the role assignment, you must first set at least the “role” field using `set_role_assignment_field`, then use `save_role_assignment`. If you don’t save a new role assignment, it will disappear when you select another role assignment or when the ADEdit session ends.

You can only use the `new_role_assignment` to create a role assignment if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical



## Syntax

```
new_role_assignment user|All AD users|All Unix users
```

## Abbreviation

newra

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
		Required. Specifies the user or group to assign the role to.
		This argument can be a user principal name (UPN) or a sAMAccountName if you are assigning a role to an Active Directory user or group, a UNIX user name or UID if assigning the role to a local UNIX user, or the UNIX group name if you assigning the role to a local UNIX group.
user   All AD users   All Unix users	string	To assign a role to a local UNIX account, use the following format: <code>oracle@localhost</code> To assign the role to a domain user, use the following format: <code>oracle@domain.name</code> You can also specify "All AD users" to assign a selected role to all Active Directory users or "All Unix users" to assign the selected role to all local UNIX users. This argument is not supported if the selected zone is a classic4 zone.

## Return value

This command returns nothing if it runs successfully.



## Examples

```
new_role_assignment adam.avery@acme.com
```

This example creates a new role assignment for `adam.avery@acme.com` in the current zone. You must set at least one role assignment field and an available time for the role to be effective.

The following example creates a new role assignment for the local UNIX user `oracle` in the current zone.

```
new_role_assignment oracle@localhost
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select role assignment to work with:

- `get_role_assignments` returns a Tcl list of role assignments in the current zone.
- `list_role_assignments` lists to `stdout` the role assignments in the current zone.
- `select_role_assignment` retrieves a role assignment from Active Directory and stores it in memory.

After you have a role assignment stored in memory, you can use the following commands to work with that role assignment's attributes, delete the role assignment, or save information for the role assignment:

- `delete_role_assignment` deletes the selected role assignment from Active Directory and from memory.
- `get_role_assignment_field` reads a field value from the currently selected role assignment.
- `save_role_assignment` saves the selected role assignment with its current settings to Active Directory.
- `set_role_assignment_field` sets a field value in the currently selected role assignment.

.....

## new\_rs\_command

Use the `new_rs_command` command to create a new restricted shell command under the currently selected restricted shell environment and set the new restricted shell command as the currently selected restricted shell command in memory. The `umask` field for the new restricted shell command is set to a default value of 077 and default priority field (`pri`) is set to 0. For more information about restricted shell command fields, see the command description for [get\\_rsc\\_field](#).

The `new_rs_command` command does not save the new restricted shell command to Active Directory. To store the new restricted shell command in Active Directory, you must use [save\\_rs\\_command](#). If you don't save a new restricted shell command, it will disappear when you select another restricted shell command or when the ADEdit session ends.

You can only use the `new_rs_command` command if the currently selected zone is a classic4 zone. The command does not work in other types of zones.

## Zone type

Classic only

## Syntax

```
new_rs_command name
```

## Abbreviation

```
newrsc
```

## Options

This command takes no options.

.....

## Arguments

This command takes the following argument:

Argument	Type	Description
name	string	Required. Specifies the name to assign to the new restricted shell command.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
new_rs_command rsc1
```

This example creates a new restricted shell command named `rsc1` in the current zone.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the restricted shell command to work with:

- `get_rs_commands` returns a Tcl list of restricted shell commands in the current zone.
- `list_rs_commands` lists to `stdout` the restricted shell commands in the current zone.
- `select_rs_command` retrieves a restricted shell command from Active Directory and stores it in memory.

After you have a restricted shell command stored in memory, you can use the following commands to work with that restricted shell:

- `delete_rs_command` deletes the selected command from Active Directory and from memory.



- `get_rsc_field` reads a field value from the currently selected command.
- `save_rs_command` saves the selected command with its current settings to Active Directory.
- `set_rsc_field` sets a field value in the currently selected command.

## `new_rs_env`

Use the `new_rs_env` command to create a new restricted shell environment for the current zone and set the new restricted shell environment as the currently selected restricted shell environment stored in memory. The new restricted shell environment has no field values set.

The `new_rs_env` command does not save the new restricted shell environment to Active Directory. To save the new restricted shell environment to Active Directory, you must use the `save_rs_env` command. If you don't save a new restricted shell environment, it will disappear when you select another restricted shell environment or when the ADEdit session ends.

You can only use the `new_rs_env` command if the currently selected zone is a classic4 zone. The command does not work in other types of zones.

## Zone type

Classic only

## Syntax

```
new_rs_env name
```

## Abbreviation

```
newrse
```

.....

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
name	string	Required. Specifies the name to assign to the new restricted shell environment.

## Return value

This command creates a new restricted shell environment in the currently selected zone.

## Examples

```
new_rs_envs rse3
```

This example creates a new restricted environment named `rse3` in the current zone.

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with restricted shell environments:

- `get_rs_envs` returns a Tcl list of restricted shell environments.
- `list_rs_envs` lists to `stdout` the restricted shell environments.
- `select_rs_env` retrieves a restricted shell environment from Active Directory and stores it in memory.



After you have a restricted shell environment stored in memory, you can use the following commands to work with its fields:

- `delete_rs_env` deletes the current restricted shell environment from Active Directory and from memory.
- `get_rse_field` reads a field value from the current restricted shell environment.
- `save_rs_env` saves the restricted shell environment to Active Directory.
- `set_rse_field` sets a field value in the current restricted shell environment.

## new\_zone\_computer

Use the `new_zone_computer` command to create a new zone computer in the current zone and set the new zone computer as the currently selected zone computer in memory. The new zone computer has no field values set.

The `new_zone_computer` command does *not* save the new zone computer to Active Directory. To save the new zone computer, you must use `save_zone_computer`. If you don't save a new zone computer, it will disappear when you select another zone computer or when the ADEdit session ends.

The `new_zone_computer` command requires you to specify an Active Directory computer account name. If the computer name you specify is not found in Active Directory, the command does not create the zone computer.

## Zone type

Classic and hierarchical

## Syntax

```
new_zone_computer sAMAccountName@domain
```

.....

## Abbreviation

`newzc`

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>sAMAccountName @domain</code>	string	Required. Specifies the <code>sAMAccountName</code> of an Active Directory computer followed by <code>@</code> and the domain name where the computer is located.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
new_zone_computer sales2$@acme.com
```

This example creates a new zone computer `sales2@acme.com` in the current zone. Note that Tcl syntax requires “`$@`” to represent a literal “`@`”. You could also enclose the argument in braces: `{sales2@acme.com}`.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and manage the zone computers:



- `get_zone_computers` returns a Tcl list of the Active Directory names of all zone computers in the current zone.
- `list_zone_computers` lists to `stdout` the zone computers in the current zone.
- `new_zone_computer` creates a new zone computer and stores it in memory.
- `select_zone_computer` retrieves a zone computer from Active Directory and stores it in memory.

After you have a zone computer stored in memory, you can use the following commands to work with that zone computer:

- `delete_zone_computer` deletes the zone computer from Active Directory and from memory.
- `get_zone_computer_field` reads a field value from the currently selected zone computer.
- `save_zone_computer` saves the zone computer with its current settings to Active Directory.
- `set_zone_computer_field` sets a field value in the currently selected zone computer.

## `new_zone_group`

Use the `new_zone_group` command to create a new group in the current zone that is based on an existing Active Directory group. If the command is successful, the new zone group becomes the currently selected zone group stored in memory.

The `new_zone_group` command does not set any field values or save the new zone group to Active Directory. Before you can save the new zone group, you must first set at least one field for the new zone group using the `set_zone_group_field` command. You can then save the zone group profile using the `save_zone_group` command.

**Note** If the currently selected zone is a classic zone, you must set all fields for the new zone group before saving the group profile.



If you don't save a new zone group, it will disappear when you select another zone group or end the ADEdit session.

The `new_zone_group` command requires you to specify an Active Directory group name. The command will search for the group first by the supplied UPN in the specified domain, then by the `sAMAccountName` in the specified domain, then by the supplied UPN in any bound domain. If the group name cannot be found, the new zone group is not created.

## Zone type

Classic and hierarchical

## Syntax

```
new_zone_group AD_group_UPN
```

## Abbreviation

newzg

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
AD_group_UPN	string	Required. Specifies the user principal name (UPN) of an Active Directory group.



## Return value

This command returns nothing if it runs successfully.

## Examples

```
new_zone_group poweradmins@acme.com
```

This example creates a new zone group named `poweradmins@acme.com` in the current zone.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select zone groups:

- `get_zone_groups` returns a Tcl list of the Active Directory names of all zone groups in the current zone.
- `list_zone_groups` lists to `stdout` the zone groups in the current zone.
- `select_zone_group` retrieves a zone group from Active Directory and stores it in memory.

After you have a zone group stored in memory, you can use the following commands to work with that zone group:

- `delete_zone_group` deletes the selected zone group from Active Directory and from memory.
- `get_zone_group_field` reads a field value from the current zone group.
- `save_zone_group` saves the selected zone group with its current settings to Active Directory.
- `set_zone_group_field` sets a field value in the current zone group.



## new\_zone\_user

Use the `new_zone_user` command to create a new zone user in the current zone based on an existing Active Directory user. If the command is successful, the new zone user becomes the currently selected zone user stored in memory.

The `new_zone_user` command does not set any field values or save the new zone user to Active Directory. Before you can save the new zone user, you must first set at least one field value using the `set_zone_user_field` command. You can then save the zone user profile using the `save_zone_user` command.

**Note** If the currently selected zone is a classic zone, you must set all fields for the new zone user before saving the user profile.

If you don't save a new zone user, it will disappear when you select another zone user or end the ADEdit session.

You can create more than one zone user within a zone based on a single Active Directory user. The first zone user you create uses the Active Directory user's user principal name (UPN), for example, `martin.moore@acme.com`. Any other zone users you create for the same Active Directory user must use aliases. An alias is the Active Directory user's UPN with "+n" appended where `n` is a positive integer that is unique for this Active Directory user in this zone. For example, `martin.moore@acme.com+1` is an alias, as is `martin.moore@acme.com+5`. Alias integers need not be consecutive or in order. (Note that SFU zones do not support user aliases.)

The `new_zone_user` command requires you to specify Active Directory user name. The command will search for the user first by the supplied UPN in the specified domain, then by the `sAMAccountName` in the specified domain, then by the supplied UPN in any bound domain. If the user name cannot be found, the new zone user is not created.

## Zone type

Classic and hierarchical



## Syntax

```
new_zone_user AD_user_UPN
```

## Abbreviation

```
newzu
```

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
AD_user_ UPN	string	Required. Specifies the user principal name (UPN) of an Active Directory user. If you are specifying an alias, append the UPN with "+" followed by a positive integer that is unique for this user and the zone.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
new_zone_user adam.avery@acme.com
```

This example creates a new zone user based on the Active Directory user `adam.avery@acme.com` in the current zone.



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone user:

- `get_zone_users` returns a Tcl list of the Active Directory names of all zone users in the current zone.
- `list_zone_users` lists to `stdout` the zone users and their NSS data in the current zone.
- `select_zone_user` retrieves a zone user from Active Directory and stores it in memory.

After you have a zone user stored in memory, you can use the following commands to work with that zone user:

- `delete_zone_user` deletes the selected zone user from Active Directory and from memory.
- `get_zone_user_field` reads a field value from the currently selected zone user.
- `save_zone_user` saves the selected zone user with its current settings to Active Directory.
- `set_zone_user_field` sets a field value in the currently selected zone user.

## pop

Use the `pop` command to retrieve a previously-stored context of bindings and selected objects from the top of the context stack. This command replaces the current ADEdit context with the retrieved context. Popping a context from the context stack removes the context from the stack.

This command is useful for Tcl scripts that use subroutines. A push can save the context before it's altered in the subroutine; a pop can return the saved context when the subroutine returns.

.....

## Zone type

Not applicable

## Syntax

```
pop
```

## Abbreviation

None.

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully. If the stack is empty, it returns a message stating so.

## Examples

```
pop
```

This example retrieves the context from the top of the context stack and uses it as the current AEdit context.



## Related commands

The following commands perform actions related to this command:

- `show` returns the current context of ADEdit, including its bound domains and its currently selected objects.
- `push` saves the current ADEdit context to the ADEdit context stack.

## principal\_from\_sid

Use the `principal_from_sid` command look up the security principal for a specified security identifier (SID) in Active Directory. If the security identifier is found, the command returns the Active Directory name of the principal.

## Zone type

Not applicable

## Syntax

```
principal_from_sid [-upn] sid
```

## Abbreviation

`pfs`

## Options

This command takes the following option:

Option	Description
<code>-upn</code>	Returns the user names in user principal name (UPN) format, not the default <code>SAMAccount@domain</code> format.



## Arguments

This command takes the following argument:

Argument	Type	Description
sid	string	Required. Specifies the security identifier of an Active Directory security principal.

## Return value

This command returns the Active Directory name of the principal if it finds a principal. If it does not find a principal, it returns a message stating so.

## Examples

```
principal_from_sid S-1-5-21-2076040321-3326545908-468068287-1159
```

This example returns the principal name: `oracle_machines@acme.com`

## Related commands

The following commands perform actions related to this command:

- `principal_to_dn` searches Active Directory for a user principal name (UPN) and, if found, returns the corresponding distinguished name (DN).
- `dn_to_principal` searches Active Directory for a distinguished name (DN) and, if found, returns the corresponding user principal name (UPN).

## principal\_to\_dn

Use the `principal_to_dn` command to search Active Directory for the specified user principal name (UPN) of a security principal (user, machine, or group). If a security principal is found for the specified UPN, the command returns the distinguished name (DN) of the principal.

.....

## Zone type

Not applicable

## Syntax

```
principal_to_dn principal_upn
```

## Abbreviation

ptd

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
principal_upn	string	Required. Specifies the user principal name (UPN) of a security principal.

## Return value

This command returns a distinguished name. If the command doesn't find the specified security principal in Active Directory, it presents a message that it didn't find the principal.

## Examples

```
principal_to_dn brenda.butler@acme.com
```

• • • • •

This example returns the distinguished name for the specified UPN:

```
cn=brenda butler,cn=users,dc=acme,dc=com
```

## Related commands

The following commands perform actions related to this command:

- `dn_from_domain` converts a domain's dotted name to a distinguished name.
- `get_parent_dn` returns the parent of an LDAP path as a distinguished name.
- `get_rdn` returns the relative distinguished name of an LDAP path.
- `dn_to_principal` searches Active Directory for a distinguished name, and, if found, returns the corresponding user principal name (UPN).
- `principal_from_sid` searches Active Directory for a security identifier and returns the security principal associated with the security identifier.

## principal\_to\_id

Use the `principal_to_id` command to search Active Directory for the specified user principal name (UPN) of a user or group security principal. If a security principal is found for the specified UPN, the command returns the numeric identifier for the principal.

## Zone type

Not applicable

## Syntax

```
principal_to_id [-apple] upn
```



## Abbreviation

`pti`

## Options

This command takes the following option:

Option	Description
<code>-apple</code>	Specifies that you want to use the Apple scheme for generating the UID or GID for the specified user or group principal. If you don't specify this option, the UID or GID returned is based on the Centrify Auto Zone scheme.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>upn</code>	string	Required. Specifies the user principal name (UPN) of a user or group security principal.

## Return value

This command returns a unique UID or GID based on either the Apple methodology or the Centrify Auto Zone methodology for generating numeric identifiers. If the user or group principal is not found in Active Directory, the command returns an error message indicating that it didn't find the principal.

## Examples

```
principal_to_id -apple brenda.butler@acme.com
```

This example returns the UID for the specified user generated using the Apple scheme:

## Related commands

The following commands perform actions related to this command:

- `guid_to_id` accepts a globally unique identifier (GUID) for a user or group and returns a UID or GID generated using the Apple scheme.
- `principal_from_sid` searches Active Directory for a security identifier and returns the security principal associated with the security identifier.

## push

Use the `push` command to save the current ADEdit context—its bindings and selected objects in memory—to a context stack. This command leaves the current context in place, so all current bindings and selected objects remain in effect in ADEdit after the push.

This command is useful for Tcl scripts that use subroutines. You can use the `push` command to save the context before it's altered in the subroutine. You can then use the `pop` command to retrieve the saved context when the subroutine returns.

## Zone type

Not applicable

## Syntax

```
push
```

## Abbreviation

None.

.....

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing.

## Examples

```
push
```

The example saves the current ADEdit context.

## Related commands

The following commands perform actions related to this command:

- `show` returns the current context of ADEdit, including its bound domains and currently selected objects.
- `pop` restores the context from the top of the ADEdit context stack to ADEdit.

## quit

Use the `quit` command to quit ADEdit and return to the shell from which ADEdit was launched. You can also end an interactive ADEdit session by pressing `ctrl-D` or entering `exit`.



**Note** If you enter the `exit` command, understand that it will terminate the session immediately without performing a commit operation.

## Zone type

Not applicable

## Syntax

```
quit
```

## Abbreviation

```
q
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing.

## Examples

```
quit
```

This example ends an ADEdit session.

.....

## Related commands

None.

## remove\_command\_from\_role

Use the `remove_command_from_role` command to remove a UNIX command from the currently selected role stored in memory.

The `remove_command_from_role` command does not change the role as it is stored in Active Directory. You must save the role before the removed command takes effect in Active Directory. If you select another role or quit ADEdit before saving the role, any UNIX commands you have removed since the last save won't take effect.

You can only use the `remove_command_from_role` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
remove_command_from_role command[/zonename]
```

## Abbreviation

`rcfr`

## Options

This command takes no options.

.....

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>command[ /zonename ]</code>	string	Required. Specifies the name of a UNIX command to remove from the currently selected role.  If the UNIX command that you want to remove is defined in the current zone, the <i>zonename</i> argument is optional. If the UNIX command right is defined in a zone other than the currently selected zone, the <i>zonename</i> argument is required to identify the specific command to remove.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
remove_command_from_role basicshell/global
```

This example removes the UNIX command named `basicshell`, which is defined in the `global` zone, from the currently selected role.

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with:

- `get_roles` returns a Tcl list of roles in the current zone.
- `list_roles` lists to `stdout` the roles in the current zone.
- `new_role` creates a new role and stores it in memory.
- `select_role` retrieves a role from Active Directory and stores it in memory.



After you have a role stored in memory, you can use the following commands to work with that role:

- `add_command_to_role` adds a UNIX command to the current role.
- `add_pamapp_to_role` adds a PAM application to the current role.
- `delete_role` deletes the selected role from Active Directory and from memory.
- `get_role_apps` returns a Tcl list of the PAM applications associated with the current role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the current role.
- `list_role_rights` returns a list of all UNIX commands and PAM applications associated with the current role.
- `remove_pamapp_from_role` removes a PAM application from the current role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the current role.

## remove\_object\_value

Use the `remove_object_value` command to remove a value from a multi-valued attribute of a specified Active Directory object. This command only affects the specified attribute for specified object in Active Directory. The command does not change the currently selected Active Directory object in memory, if there is one.

If the field or value to be removed isn't valid, Active Directory will report an error and `remove_object_value` won't remove the value.

This command is useful for fields that may be very large—members of a group, for example.

## Zone type

Not applicable



## Syntax

```
remove_object_value dn field value
```

## Abbreviation

rov

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
dn	string	Required. Specifies the distinguished name (DN) of the Active Directory object from which to remove a value.
		Required. Specifies the name of a multi-valued attribute in the currently selected Active Directory object from which to remove the value.
field	string	This argument can be any field that is valid for the type of the Active Directory object you have specified using the <i>dn</i> argument. For example, if the Active Directory object specified is a computer object, the <i>field</i> argument might be <code>operatingSystem</code> .
value		Required. Specifies the value to remove from the field. The data type of the <i>value</i> depends on the <i>field</i> you specify.

## Return value

This command returns nothing if it runs successfully.



## Examples

```
remove_object_value cn=groups,dc=acme,dc=com users  
adam.avery
```

This example removes the value `adam.avery` from the `users` field of the `groups` object in Active Directory.

## Related commands

The following commands enable you to view and select the object to work with:

- `get_objects` performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of objects matching the search criteria.
- `new_object` creates a new Active Directory object and stores it in memory.
- `select_object` retrieves an object and its attributes from Active Directory and stores it in memory.

After you have an Active Directory object stored in memory, you can use the following commands to work with that object's attributes, delete the object, or save information for the object:

- `add_object_value` adds a value to a multi-valued field attribute of the currently selected Active Directory object.
- `delete_object` deletes the selected Active Directory object from Active Directory and from memory.
- `delete_sub_tree` deletes an Active Directory object and all of its children from Active Directory.
- `get_object_field` reads a field value from the currently selected Active Directory object.
- `save_object` saves the selected Active Directory object with its current settings to Active Directory.
- `set_object_field` sets a field value in the currently selected Active Directory object.

.....

## remove\_pamapp\_from\_role

Use the `remove_pamapp_from_role` command to remove a PAM application access right from the currently selected role stored in memory.

The `remove_pamapp_from_role` command does not change the role as it is stored Active Directory. To remove the PAM application right from the role stored in Active Directory, you must save your changes using the `save_role` command. If you select another role or quit ADEdit before saving the role, any PAM applications you've removed since the last save won't take effect.

You can only use the `remove_pamapp_from_role` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
remove_pamapp_from_role app[/zonename]
```

## Abbreviation

rpamfr

## Options

This command takes no options.

## Arguments

This command takes the following argument:

---

Argument	Type	Description
		Required. Specifies the name of a PAM application right to remove from the currently selected role.
app[ /zonename ]	string	If the PAM application right that you want to remove is defined in the current zone, the <i>zonename</i> argument is optional. If the PAM application right is defined in a zone other than the currently selected zone, the <i>zonename</i> argument is required to identify the specific PAM application right to remove.

---

## Return value

This command returns nothing if it runs successfully.

## Examples

```
remove_pamapp_from_role ftp-all
```

This example removes the PAM application right named `ftp-all` defined in the currently selected zone from the currently selected role.

To remove the PAM application right when it is defined in the `seattle` zone, you would include the zone name:

```
remove_pamapp_from_role ftp-all/seattle
```

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with:

- `get_roles` returns a Tcl list of roles in the current zone.
- `list_roles` lists to `stdout` the roles in the current zone.
- `new_role` creates a new role and stores it in memory.
- `select_role` retrieves a role from Active Directory and stores it in memory.



After you have a role stored in memory, you can use the following commands to work with that role:

- `add_command_to_role` adds a UNIX command to the current role.
- `add_pamapp_to_role` adds a PAM application to the current role.
- `delete_role` deletes the selected role from Active Directory and from memory.
- `get_role_apps` returns a Tcl list of the PAM applications associated with the current role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the current role.
- `list_role_rights` returns a list of all UNIX commands and PAM applications associated with the current role.
- `remove_command_from_role` removes a UNIX command from the current role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the current role.

## remove\_sd\_ace

Use the `remove_sd_ace` command to remove an access control entry (ACE) in ACE string form from a security descriptor (SD) in SDDL (security descriptor description language) form.

The command looks for the supplied ACE string within the supplied SDDL string. If the command finds the ACE string, it removes it from the SDDL string and returns the SDDL string.

## Zone type

Not applicable

.....

## Syntax

```
remove_sd_ace sddl_string ace_string
```

## Abbreviation

rsa

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
sddl_string	string	Required. Specifies a security descriptor in SDDL format.
ace_string	string	Required. Specifies an access control entry in ACE string form, which is always enclosed in parentheses.

## Return value

This command returns a modified security descriptor in SDDL format if it runs successfully.

## Examples

This example removes the first ACE string from an SDDL. The ACE string to remove is at the end of the command

```
(A;;SDRCWDWOCCDCLCSWRPWPDTLOCR;;;SY):
```

.....

```
remove_sd_ace O:DAG:DAD:AI
(A;;SDRCWDWOCCDCLCSWRPWPDTLOCR;;;SY)
(OA;;RCWDWOCCDCLCSWRPWPLOCR;;;DA) (OA;;CCDC;bf967aba-0de6-11d0-a285-00aa003049e2;;;AO) (OA;;CCDC;bf967a9c-0de6-11d0-a285-00aa003049e2;;;AO) (OA;;CCDC;bf967aa8-0de6-11d0-a285-00aa003049e2;;;PO) (A;;RCLCRPLO;;;AU) (OA;;CCDC;4828cc14-1437-45bc-9b07-ad6f015e5f28;;;AO) (OA;CIIOID;RP;4c164200-20c0-11d0-a768-00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU) (OA;CIIOID;RP;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU) (OA;CIIOID;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU) (OA;CIIOID;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU) (OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU) (OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU) (OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU) (OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU) (OA;CIIOID;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU) (OA;CIIOID;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU) (OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-00a0c983f608;bf967a86-0de6-11d0-a285-00aa003049e2;ED) (OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-00a0c983f608;bf967a9c-0de6-11d0-a285-00aa003049e2;ED) (OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-00a0c983f608;bf967aba-0de6-11d0-a285-00aa003049e2;ED) (OA;CIIOID;RCLCRPLO;;;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU) (OA;CIIOID;RCLCRPLO;;;bf967a9c-0de6-11d0-a285-00aa003049e2;RU) (OA;CIIOID;RCLCRPLO;;;bf967aba-0de6-11d0-a285-00aa003049e2;RU) (OA;CIID;RPWPCR;91e647de-d96f-4b70-9557-d63ff4f3ccd8;;;PS)
(A;CIID;SDRCWDWOCCDCLCSWRPWPDTLOCR;;;EA) (A;CIID;LC;;;RU)
(A;CIID;SDRCWDWOCCCLCSWRPWPLOCR;;;BA)
(A;;SDRCWDWOCCDCLCSWRPWPDTLOCR;;;SY)
```

The command returns the SDDL string without the first ACE string:

```
O:DAG:DAD:AI (A;;RCWDWOCCDCLCSWRPWPLOCR;;;DA)
(OA;;CCDC;bf967aba-0de6-11d0-a285-00aa003049e2;;;AO)
(OA;;CCDC;bf967a9c-0de6-11d0-a285-00aa003049e2;;;AO)
```

.....

```
(OA;;CCDC;bf967aa8-0de6-11d0-a285-00aa003049e2;;PO)
(A;;RCLCRPLO;;;AU) (OA;;CCDC;4828cc14-1437-45bc-9b07-
ad6f015e5f28;;AO) (OA;CIIOID;RP;4c164200-20c0-11d0-a768-
00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;4c164200-20c0-11d0-a768-
00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a86-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a9c-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967aba-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RCLCRPLO;;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RCLCRPLO;;bf967a9c-0de6-11d0-
a285-00aa003049e2;RU) (OA;CIIOID;RCLCRPLO;;bf967aba-0de6-
11d0-a285-00aa003049e2;RU) (OA;CIID;RPWPCR;91e647de-d96f-
4b70-9557-d63ff4f3ccd8;;PS)
(A;CIID;SDRCWDWOCDCCLCSWRPWPDTLOCR;;;EA) (A;CIID;LC;;;RU)
(A;CIID;SDRCWDWOCCLCSWRPWPLOCR;;;BA)
(A;;SDRCWDWOCDCCLCSWRPWPDTLOCR;;;SY)
```

## Related commands

The following commands enable you to work with security descriptor strings:

- `add_sd_ace` adds an access control entry to a security descriptor.
- `explain_sd` converts an SD in SDDL format to a human-readable form.

.....

- `set_sd_owner` sets the owner of a security descriptor.

## rename\_object

Use the `rename_object` command to rename the selected object. You can replace only the first relative distinguished name in the selected object. You do not need to save the object after you change the name.

## Zone type

Not applicable

## Syntax

```
rename_object name
```

## Abbreviation

rno

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
name	string	Required. Specifies the replacement relative distinguished name for the first relative distinguished name in the selected object.

.....

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example selects the user object `Lois Lane` and changes her name to `LoisLane`:

```
select_object "cn=Lois Lane,cn=Users,dc=demo,dc=test"  
rename_object LoisLane
```

The following example selects the organizational unit `unixServers` and renames it to `UNIX Servers`:

```
select_object "ou=UnixServers,ou=Centrify,dc=demo,dc=test"  
rno "UNIX Servers"
```

In both examples, quotes are required to preserve spaces in object names.

## Related commands

The following command performs actions related to this command:

- `select_object` selects the object you want to rename.

## save\_dz\_command

Use the `save_dz_command` command to save the currently selected UNIX command stored in memory to Active Directory. You must save a UNIX command for any changes you make using ADEdit to take effect in Active Directory. If you select another UNIX command or end the ADEdit session before saving the currently selected UNIX command, your changes will be lost.

## Zone type

Classic and hierarchical



## Syntax

```
save_dz_command
```

## Abbreviation

```
svdzc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_dz_command
```

This example saves the currently selected UNIX command to Active Directory.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a UNIX command:



- `get_dz_commands` returns a Tcl list of UNIX commands in the current zone.
- `list_dz_commands` lists to `stdout` the UNIX commands in the current zone.
- `new_dz_command` creates a new UNIX command and stores it in memory.
- `select_dz_command` retrieves a UNIX command from Active Directory and stores it in memory.

After you have a UNIX command stored in memory, you can use the following commands to work with that command:

- `delete_dz_command` deletes the selected command from Active Directory and from memory.
- `get_dzc_field` reads a field value from the currently selected command.
- `set_dzc_field` sets a field value in the currently selected command.

## save\_local\_group\_profile

Use the `save_local_group_profile` command to save the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.

Whenever you execute the `new_local_group_profile` or `select_local_group_profile` command, the group continues to be selected until you execute the `save_local_group_profile` command.

You can save a group object before the group profile is complete. However, the group profile is not added to `/etc/group` on each UNIX and Linux computer in the zone until the group profile is complete and the `profileflag` field is set to 1 (enabled). See `new_local_group_profile` for details about which attributes a group profile must have to be considered complete.

## Zone type

Hierarchical only.

.....

## Syntax

```
save_local_group_profile
```

## Abbreviation

```
svlgp
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example saves the currently selected object for the local UNIX or Linux group in the currently selected zone.

```
save_local_group_profile
```

For example, earlier you might have executed the following command to select the `marketing` group object so that you could edit its profile fields:

```
select_local_group_profile marketing
```

Executing the following command would save any changes you had made to the `marketing` group object:

```
save_local_group_profile
```

## Related commands

The following related ADEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- `delete_local_group_profile` deletes a local UNIX or Linux group that has a profile defined in the current zone.
- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.



- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## save\_local\_user\_profile

Use the `save_local_user_profile` command to save the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.

Whenever you execute the `new_local_user_profile` or `select_local_user_profile` command, the user continues to be selected until you execute the `save_local_user_profile` command.

You can save a user object before the user profile is complete. However, the user profile is not added to `/etc/passwd` on each UNIX and Linux computer in the zone until the user profile is complete, the `profileflag` field is set to 1 (enabled) or 2 (disabled), and the user is assigned a visible role such as `Local` `Trusted`. See [new\\_local\\_user\\_profile](#) for details about which attributes a user profile must have to be considered complete.

## Zone type

Hierarchical only.

## Syntax

```
save_local_user_profile
```

## Abbreviation

```
svlup
```

.....

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example saves the currently selected object for the local UNIX or Linux user in the currently selected zone.

```
save_local_user_profile
```

For example, earlier you might have executed the following command to select the object for UNIX user `anton.splieth` so that you could edit its profile fields:

```
select_local_user_profile anton.splieth
```

Executing the following command would save any changes you had made to the user object for `anton.splieth`:

```
save_local_user_profile
```

## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:



- `delete_local_group_profile` deletes a local UNIX or Linux group that has a profile defined in the current zone.
- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.



- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## save\_nis\_map

Use the `save_nis_map` command to save the currently selected NIS map stored in memory to Active Directory. You must save the NIS map for any changes you make using ADEdit to take effect in Active Directory. If you select another NIS map or end the ADEdit session before saving the currently selected NIS map, your changes will be lost.

### Zone type

Not applicable

### Syntax

```
save_nis_map
```

### Abbreviation

svnm

### Options

This command takes no options.

### Arguments

This command takes no arguments.



## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_nis_map
```

This example saves the currently selected NIS map to Active Directory.

## Related commands

Before you use this command, you must have a currently selected NIS map stored in memory. The following commands enable you to view and select a NIS map:

- [get\\_nis\\_maps](#) returns a Tcl list of NIS maps in the current zone.
- [list\\_nis\\_maps](#) lists to stdout the NIS maps in the current zone.
- [new\\_nis\\_map](#) creates a new NIS map and stores it in memory.
- [select\\_nis\\_map](#) retrieves a NIS map from Active Directory and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map:

- [add\\_map\\_entry](#) or [add\\_map\\_entry\\_with\\_comment](#) adds a map entry to the currently selected NIS map.
- [delete\\_map\\_entry](#) removes an entry from the currently selected NIS map.
- [delete\\_nis\\_map](#) deletes the selected NIS map from Active Directory and from memory.
- [get\\_nis\\_map](#) or [get\\_nis\\_map\\_with\\_comment](#) returns a Tcl list of the map entries in the currently selected NIS map.
- [get\\_nis\\_map\\_field](#) reads a field value from the currently selected NIS map.



- `list_nis_map` or `list_nis_map_with_comment` lists to `stdout` the map entries in the currently selected NIS map.

## save\_object

Use the `save_object` command to save the currently selected Active Directory object stored in memory to Active Directory. You must save the Active Directory object for any changes you make using ADEdit to take effect in Active Directory. If you select another Active Directory object or end the ADEdit session before saving the currently selected object, your changes will be lost.

If an object has invalid attributes or values or is the wrong class for the container where it's being saved, Active Directory will report an error and the object will not be saved.

## Zone type

Not applicable

## Syntax

```
save_object
```

## Abbreviation

svo

## Options

This command takes no options.

.....

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_object
```

This example saves the currently selected Active Directory object to Active Directory.

## Related commands

The following commands enable you to view and select the object to work with:

- [get\\_objects](#) performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of objects matching the specified search criteria.
- [new\\_object](#) creates a new Active Directory object and stores it in memory.
- [select\\_object](#) retrieves an object and its attributes from Active Directory and stores it in memory.

After you have an Active Directory object stored in memory, you can use the following commands to work with that object's attributes, delete the object, or save information for the object:

- [add\\_object\\_value](#) adds a value to a multi-valued field attribute of the currently selected Active Directory object.
- [delete\\_object](#) deletes the selected Active Directory object from Active Directory and from memory.



- `delete_sub_tree` deletes an Active Directory object and all of its children from Active Directory.
- `get_object_field` reads a field value from the currently selected Active Directory object.
- `remove_object_value` removes a value from a multi-valued field attribute of the currently selected Active Directory object.
- `set_object_field` sets a field value in the currently selected Active Directory object.

## save\_pam\_app

Use the `save_pam_app` command to save the currently selected PAM application access right stored in memory to Active Directory. You must save the PAM application right for any changes you make using ADEdit to take effect in Active Directory. If you select another PAM application right or end the ADEdit session before saving the currently selected PAM application right, your changes will be lost.

## Zone type

Classic and hierarchical

## Syntax

```
save_pam_app
```

## Abbreviation

```
svpam
```

## Options

This command takes no options.

.....

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_pam_app
```

This example saves the currently selected PAM application to Active Directory.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a PAM application object:

- `get_pam_apps` returns a Tcl list of PAM applications in the current zone.
- `list_pam_apps` lists to `stdout` the PAM application rights in the current zone.
- `new_pam_app` creates a new PAM application right and stores it in memory.
- `select_pam_app` retrieves a PAM application right from Active Directory and stores it in memory.

After you have a PAM application right stored in memory, you can use the following commands to work with that PAM application:

- `delete_pam_app` deletes the selected PAM application from Active Directory and from memory.
- `get_pam_field` reads a field value from the currently selected PAM application.
- `set_pam_field` sets a field value in the currently selected PAM application.



## save\_role

Use the `save_role` command to save the currently selected role stored in memory to Active Directory. You must save the role for any changes you make using ADEdit to take effect in Active Directory. If you select another role or end the ADEdit session before saving the currently selected role, your changes will be lost.

## Zone type

Classic and hierarchical

## Syntax

```
save_role
```

## Abbreviation

```
svr
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.



## Examples

```
save_role
```

This example saves the currently selected role to Active Directory.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select roles:

- `get_roles` returns a Tcl list of roles in the current zone.
- `list_roles` lists to `stdout` the roles in the current zone.
- `new_role` creates a new role and stores it in memory.
- `select_role` retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with that role:

- `add_command_to_role` adds a UNIX command to the current role.
- `add_pamapp_to_role` adds a PAM application right to the current role.
- `delete_role` deletes the selected role from Active Directory and from memory.
- `get_role_apps` returns a Tcl list of the PAM application rights associated with the current role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the current role.
- `get_role_field` reads a field value from the current role.
- `list_role_rights` returns a list of all UNIX commands and PAM application rights associated with the current role.
- `remove_command_from_role` removes a UNIX command from the current role.



- `remove_pamapp_from_role` removes a PAM application right from the current role.
- `set_role_field` sets a field value in the current role.

## save\_role\_assignment

Use the `save_role_assignment` command to save the currently selected role assignment stored in memory to Active Directory. You must save the role assignment for any changes you make using ADEdit to take effect in Active Directory. If you select another role assignment or end the ADEdit session before saving the currently selected role assignment, your changes will be lost.

## Zone type

Classic and hierarchical

## Syntax

```
save_role_assignment
```

## Abbreviation

```
svra
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.



## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_role_assignment
```

This example saves the currently selected role assignment to Active Directory.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select role assignment to work with:

- [get\\_role\\_assignments](#) returns a Tcl list of role assignments in the current zone.
- [list\\_role\\_assignments](#) lists to `stdout` the role assignments in the current zone.
- [new\\_role\\_assignment](#) creates a new role assignment and stores it in memory.
- [select\\_role\\_assignment](#) retrieves a role assignment from Active Directory and stores it in memory.

After you have a role assignment stored in memory, you can use the following commands to work with that role assignment's attributes, delete the role assignment, or save information for the role assignment:

- [delete\\_role\\_assignment](#) deletes the selected role assignment from Active Directory and from memory.
- [get\\_role\\_assignment\\_field](#) reads a field value from the current role assignment.
- [save\\_role\\_assignment](#) saves the selected role assignment with its current settings to Active Directory.
- [set\\_role\\_assignment\\_field](#) sets a field value in the current role assignment.

.....

## save\_rs\_command

Use the `save_rs_command` command to save the currently selected restricted shell command that is stored in memory to Active Directory. You must save the restricted shell command for any changes you make using ADEdit to take effect in Active Directory. If you select another restricted shell command or end the ADEdit session before saving the currently selected restricted shell command, your changes will be lost.

## Zone type

Classic only

## Syntax

```
save_rs_command
```

## Abbreviation

```
svrsc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.



## Examples

`save_rs_command`

This example saves the currently selected RSC to Active Directory.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the restricted shell command to work with:

- `get_rs_commands` returns a Tcl list of restricted shell commands in the current zone.
- `list_rs_commands` lists to `stdout` the restricted shell commands in the current zone.
- `new_rs_command` creates a new restricted shell command and stores it in memory.
- `select_rs_command` retrieves a restricted shell command from Active Directory and stores it in memory.

After you have a restricted shell command stored in memory, you can use the following commands to work with that restricted shell:

- `delete_rs_command` deletes the selected command from Active Directory and from memory.
- `get_rsc_field` reads a field value from the currently selected command.
- `set_rsc_field` sets a field value in the currently selected command.

## `save_rs_env`

Use the `save_rs_env` command to save the currently selected restricted shell environment that is stored in memory to Active Directory. You must save the selected restricted shell environment for any changes you make using ADEdit to take effect in Active Directory. If you select another restricted shell environment or end the ADEdit session before saving the currently selected restricted shell environment, your changes will be lost.

.....

## Zone type

Classic only

## Syntax

```
save_rs_env
```

## Abbreviation

```
svrse
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_rs_env
```

This command saves the currently selected restricted shell environment to Active Directory.



## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with restricted shell environments:

- [get\\_rs\\_envs](#) returns a Tcl list of restricted shell environments.
- [list\\_rs\\_envs](#) lists to stdout the restricted shell environments.
- [new\\_rs\\_env](#) creates a new restricted shell environment and stores it in memory.
- [select\\_rs\\_env](#) retrieves a restricted shell environment from Active Directory and stores it in memory.

After you have a restricted shell environment stored in memory, you can use the following commands to work with its fields:

- [delete\\_rs\\_env](#) deletes the current restricted shell environment from Active Directory and from memory.
- [get\\_rse\\_field](#) reads a field value from the current restricted shell environment.
- [set\\_rse\\_field](#) sets a field value in the current restricted shell environment.

## save\_zone

Use the `save_zone` command to save the currently selected zone stored in memory to Active Directory. You must save the selected zone for any changes you make using ADEdit to take effect in Active Directory. If you select another zone or end the ADEdit session before saving the currently selected zone, your changes will be lost.

This command only saves fields that are properties in the currently selected zone. The command does not save any users or groups added to a zone. You must save users and groups individually using the [save\\_zone\\_user](#) and [save\\_zone\\_group](#) commands.

.....

## Zone type

Classic and hierarchical

## Syntax

```
save_zone
```

## Abbreviation

```
svz
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_zone
```

This example saves the currently selected zone or computer role to Active Directory.



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone to work with:

- `create_zone` creates a new zone in Active Directory.
- `get_zones` returns a Tcl list of all zones within a specified domain.
- `select_zone` retrieves a zone from Active Directory and stores it in memory.

After you have a zone stored in memory, you can use the following commands to work with that zone:

- `delegate_zone_right` delegates a zone use right to a specified user or computer.
- `delete_zone` deletes the selected zone from Active Directory and memory.
- `get_child_zones` returns a Tcl list of child zones, computer roles, or computer zones.
- `get_zone_field` reads a field value from the currently selected zone.
- `get_zone_nss_vars` returns the NSS substitution variable for the selected zone.
- `set_zone_field` sets a field value in the currently selected zone.

## `save_zone_computer`

Use the `save_zone_computer` command to save the currently selected zone computer stored in memory to Active Directory. You must set at least one field value before you can save a zone computer. In classic zones, you must set all field values before you can save a zone computer.

You must save the selected zone computer for any changes you make using ADEdit to take effect in Active Directory. If you select another zone computer or end the ADEdit session before saving the currently selected zone computer, your changes will be lost.

.....

## Zone type

Classic and hierarchical

## Syntax

```
save_zone_computer
```

## Abbreviation

```
svzc
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_zone_computer
```

This example saves the currently selected zone computer to Active Directory.



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and manage the zone computers:

- `get_zone_computers` returns a Tcl list of the Active Directory names of all zone computers in the current zone.
- `list_zone_computers` lists to `stdout` the zone computers in the current zone.
- `new_zone_computer` creates a new zone computer and stores it in memory.
- `select_zone_computer` retrieves a zone computer from Active Directory and stores it in memory.

After you have a zone computer stored in memory, you can use the following commands to work with that zone computer:

- `delete_zone_computer` deletes the zone computer from Active Directory and from memory.
- `get_zone_computer_field` reads a field value from the currently selected zone computer.
- `save_zone_computer` saves the zone computer with its current settings to Active Directory.
- `set_zone_computer_field` sets a field value in the currently selected zone computer.

## save\_zone\_group

Use the `save_zone_group` command to save the currently selected zone group stored in memory to Active Directory. You must set at least one field value before you can save a zone group. In classic zones, you must set all field values before you can save a zone group.

You must save the selected zone group for any changes you make using ADEdit to take effect in Active Directory. If you select another zone group or end the ADEdit session before saving the currently selected zone group, your changes will be lost.

.....

## Zone type

Classic and hierarchical

## Syntax

```
save_zone_group
```

## Abbreviation

```
svzg
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_zone_group
```

This example saves the currently selected zone group to Active Directory.



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select zone groups:

- `get_zone_groups` returns a Tcl list of the Active Directory names of all zone groups in the current zone.
- `list_zone_groups` lists to stdout the zone groups in the current zone.
- `new_zone_group` creates a new zone group and stores it in memory.
- `select_zone_group` retrieves a zone group from Active Directory and stores it in memory.

After you have a zone group stored in memory, you can use the following commands to work with that zone group:

- `delete_zone_group` deletes the selected zone group from Active Directory and from memory.
- `get_zone_group_field` reads a field value from the currently selected zone group.
- `save_zone_group` saves the selected zone group with its current settings to Active Directory.
- `set_zone_group_field` sets a field value in the currently selected zone group.

## `save_zone_user`

Use the `save_zone_user` command to save the currently selected zone user stored in memory to Active Directory. You must set at least one field value before you can save a zone user. In classic zones, you must set all field values before you can save a zone user.

You must save the selected zone user for any changes you make using ADEdit to take effect in Active Directory. If you select another zone user or end the ADEdit session before saving the currently selected zone user, your changes will be lost.

.....

## Zone type

Classic and hierarchical

## Syntax

```
save_zone_user
```

## Abbreviation

```
svzu
```

## Options

This command takes no options.

## Arguments

This command takes no arguments.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
save_zone_user
```

This example saves the currently selected zone user to Active Directory.



## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone user:

- `get_zone_users` returns a Tcl list of the Active Directory names of all zone users in the current zone.
- `list_zone_users` lists to `stdout` the zone users and their NSS data in the current zone.
- `new_zone_user` creates a new zone user and stores it in memory.
- `select_zone_user` retrieves a zone user from Active Directory and stores it in memory.

After you have a zone user stored in memory, you can use the following commands to work with that zone user:

- `delete_zone_user` deletes the selected zone user from Active Directory and from memory.
- `get_zone_user_field` reads a field value from the currently selected zone user.
- `save_zone_user` saves the selected zone user with its current settings to Active Directory.
- `set_zone_user_field` sets a field value in the currently selected zone user.

## select\_dz\_command

Use the `select_dz_command` command to retrieve a UNIX command in the currently selected zone from Active Directory. This command stores the selected UNIX command in memory, and makes it the currently selected UNIX command for subsequent ADEdit commands. The UNIX command remains selected until you select another UNIX command or zone, delete the UNIX command, or end the ADEdit session.

If you use ADEdit commands such as `set_dzc_field` to change settings for the selected UNIX command, you must save the selected UNIX command using the `save_dz_command` command for your changes to take effect in Active



Directory. If you select another UNIX command or end the ADEdit session before saving the currently selected UNIX command, your changes will be lost.

You can only use the `select_dz_command` command to select UNIX commands if the currently selected zone is a classic4 or hierarchical zone. The command does not work for other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
select_dz_command command
```

## Abbreviation

sldzc

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
<code>command</code>	string	Required. Specifies the name of the UNIX command to select.

## Return value

This command returns nothing if it runs successfully.



## Examples

```
select_dz_command account_manager
```

This example looks for the UNIX command named “account\_manager” in the current zone and, if found, selects it as the current UNIX command.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a UNIX command to work with:

- `get_dz_commands` returns a Tcl list of UNIX commands in the current zone.
- `list_dz_commands` lists to stdout the UNIX commands in the current zone.
- `new_dz_command` creates a new UNIX command and stores it in memory.

After you have a UNIX command stored in memory, you can use the following commands to work with that command:

- `delete_dz_command` deletes the selected command from Active Directory and from memory.
- `get_dzc_field` reads a field value from the currently selected command.
- `save_dz_command` saves the selected command with its current settings to Active Directory.
- `set_dzc_field` sets a field value in the currently selected command.

## select\_local\_group\_profile

Use the `select_local_group_profile` command to select a local UNIX or Linux group object for viewing or editing. The group that you specify remains selected until you execute the `save_local_group_profile` command.

.....

You typically use `select_local_group_profile` to select a group profile before you execute `get_local_group_profile_field` or `set_local_group_profile_field` to view or edit profile information.

## Zone type

Hierarchical only.

## Syntax

```
select_local_group_profile group_name
```

## Abbreviation

sllgp

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>group_name</code>	string	Required. Specifies the UNIX name of the local group to select.

## Return value

This command returns nothing if it runs successfully.



## Examples

The following example selects the object for the local UNIX or Linux group `marketing`.

```
select_local_group_profile marketing
```

## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- `delete_local_group_profile` deletes a local UNIX or Linux group that has a profile defined in the current zone.
- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.



- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## select\_local\_user\_profile

Use the `select_local_user_profile` command to select a local UNIX or Linux user object for viewing or editing. The user that you specify remains selected until you execute the `save_local_user_profile` command.

You typically use `select_local_user_profile` to select a user profile before you execute `get_local_user_profile_field` or `set_local_user_profile_field` to view or edit profile information.

## Zone type

Hierarchical only.

## Syntax

```
select_local_user_profile user_name
```

.....

## Abbreviation

sllup

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
user_name	string	Required. Specifies the UNIX name of the local user to select.

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example selects the object for the local UNIX or Linux user `anton.splieth`.

```
select_local_user_profile anton.splieth
```

## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- [delete\\_local\\_group\\_profile](#) deletes a local UNIX or Linux group that has a profile defined in the current zone.



- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

.....

## select\_nis\_map

Use the `select_nis_map` command to retrieve a NIS map in the currently selected zone from Active Directory. This command stores the NIS map in memory, and makes it the currently selected NIS map for subsequent ADEdit commands. The NIS map remains selected until you select another NIS map or zone, delete the NIS map, or end the ADEdit session.

If you use ADEdit commands such as `add_map_entry` to change settings for the selected NIS map, you must save the selected NIS map using the `save_nis_map` command for your changes to take effect in Active Directory. If you select another NIS map or end the ADEdit session before saving the currently selected NIS map, your changes will be lost.

## Zone type

Not applicable

## Syntax

```
select_nis_map map
```

## Abbreviation

slnm

## Options

This command takes no options.

## Arguments

This command takes the following arguments:



Argument	Type	Description
map	string	Required. Specifies the name of the NIS map to retrieve from Active Directory.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
select_nis_map Printers
```

This example looks for the NIS map named “Printers” in the current zone and, if found, selects it as the current NIS map.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select NIS maps:

- [get\\_nis\\_maps](#) returns a Tcl list of NIS maps in the current zone.
- [list\\_nis\\_maps](#) returns a list to stdout of all NIS maps in the current zone.
- [new\\_nis\\_map](#) creates a new NIS map and stores it in memory.

After you have a NIS map stored in memory, you can use the following commands to work with that map:

- [add\\_map\\_entry](#) or [add\\_map\\_entry\\_with\\_comment](#) adds an entry to the current NIS map stored in memory.
- [delete\\_map\\_entry](#) removes an entry from the current NIS map.
- [delete\\_nis\\_map](#) deletes the selected NIS map from Active Directory and from memory.
- [get\\_nis\\_map](#) or [get\\_nis\\_map\\_with\\_comment](#) returns a Tcl list of the map entries in the current NIS map.



- `get_nis_map_field` reads a field value from the current NIS map.
- `list_nis_map` or `list_nis_map_with_comment` lists to `stdout` the map entries in the current NIS map.
- `save_nis_map` saves the selected NIS map with its current entries to Active Directory.

## select\_object

Use the `select_object` command to retrieve the specified Active Directory object and its attributes from Active Directory. This command stores the object in memory and makes it the currently selected Active Directory object. You can use options to retrieve the rootDSE of the object or to list specific attributes to retrieve for the object.

## Zone type

Not applicable

## Syntax

```
select_object [-rootdse] [-attrs a1[,a2,...]] dn
```

## Abbreviation

slo

## Options

This command takes the following options:



Option	Description
- rootdse	Returns the rootDSE of the specified object instead of the object.
	Specifies the attributes to retrieve and store in memory.
-attrs <i>a1</i> [ <i>a2</i> ,...]	If you use this option, only the attributes you name ( <i>a1</i> , <i>a2</i> , <i>a3</i> , and so on) are retrieved. This option is useful if you want to limit the number of attributes returned or want to return attributes not normally returned by Active Directory.  If you do not use this option, ADEdit retrieves the attributes normally returned by Active Directory for the selected object type.

## Arguments

This command takes the following argument:

Argument	Type	Description
dn	DN	Required. Specifies the distinguished name (DN) of an Active Directory object.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
select_object "cn=users,dc=acme,dc=com"
```

This example returns the container object `cn=users,dc=acme,dc=com` and its attributes, and stores it in memory as the currently selected Active Directory object.

## Related commands

The following commands enable you to view and select the object to work with:



- `get_objects` performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of objects matching the specified search criteria.
- `new_object` creates a new Active Directory object and stores it in memory.

After you have an Active Directory object stored in memory, you can use the following commands to work with that object's attributes, delete the object, or save information for the object:

- `add_object_value` adds a value to a multi-valued field attribute of the currently selected Active Directory object.
- `delete_object` deletes the selected Active Directory object from Active Directory and from memory.
- `delete_sub_tree` deletes an Active Directory object and all of its children from Active Directory.
- `get_object_field` reads a field value from the currently selected Active Directory object.
- `remove_object_value` removes a value from a multi-valued field attribute of the currently selected Active Directory object.
- `save_object` saves the selected Active Directory object with its current settings to Active Directory.
- `set_object_field` sets a field value in the currently selected Active Directory object.

## `select_pam_app`

Use the `select_pam_app` command to retrieve a PAM application access right in the currently selected zone from Active Directory. This command stores the PAM application right in memory, and makes it the currently selected PAM application right for subsequent ADEdit commands. The PAM application right remains selected until you select another PAM application right or zone, delete the PAM application right, or end the ADEdit session.

If you use ADEdit commands such as `set_pam_field` to change settings for the selected PAM application right, you must save the selected PAM application right using the `save_pam_app` command for your changes to take



effect in Active Directory. If you select another PAM application right or end the ADEdit session before saving the currently selected PAM application right, your changes will be lost.

You can only use the `select_pam_app` command to select PAM applications if the currently selected zone is a classic4 or hierarchical zone. The command does not work for other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
select_pam_app name [/zonename]
```

## Abbreviation

slpam

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
name[ /zonename ]	string	Required. Specifies the name of the PAM application right to select.  If the PAM application right that you want to select is defined in the current zone, the <i>zonename</i> argument is optional.  If the PAM application right is defined in a zone other than the currently selected zone, the <i>zonename</i> argument is required to identify the specific PAM application right to select.



## Return value

This command returns nothing if it runs successfully.

## Examples

The following example retrieves the PAM application right named `sftp` in the current zone and makes it the currently selected PAM application right:

```
select_pam_app sftp
```

The following example retrieves the PAM application right named `sftp` defined in the `chicago` zone and makes it the currently selected PAM application right:

```
select_pam_app sftp/chicago
```

The definition for the PAM application right named `sftp` might be the same in both zones, but it is not required to be. Specifying the zone ensures you get the definition you expect.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. After you have a zone stored in memory, you can use the following commands to view and select the PAM application to work with:

- [get\\_pam\\_apps](#) returns a Tcl list of PAM application rights in the current zone.
- [list\\_pam\\_apps](#) lists to `stdout` the PAM application rights in the current zone.
- [new\\_pam\\_app](#) creates a new PAM application right and stores it in memory.
- [select\\_pam\\_app](#) retrieves a PAM application right from Active Directory and stores it in memory

After you have a PAM application stored in memory, you can use the following commands to work with that PAM application's attributes, delete the PAM application, or save information for the PAM application:



- `delete_pam_app` deletes the selected PAM application right from Active Directory and from memory.
- `get_pam_field` reads a field value from the currently selected PAM application right.
- `save_pam_app` saves the selected PAM application right with its current settings to Active Directory.
- `set_pam_field` sets a field value in the currently selected PAM application right.

## select\_role

Use the `select_role` command to retrieve a role in the currently selected zone from Active Directory. This command stores the role in memory, and makes it the currently selected role for subsequent ADEdit commands. The role remains selected until you select another role or zone, delete the role, or end the ADEdit session.

If you use ADEdit commands such as `set_role_field` to change settings for the selected role, you must save the selected role using the `save_role` command for your changes to take effect in Active Directory. If you select another role or end the ADEdit session before saving the currently selected role, your changes will be lost.

You can only use the `select_role` command to select roles if the currently selected zone is a classic4 or hierarchical zone. The command does not work for other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
select_role role
```

.....

## Abbreviation

slr

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
role	string	Required. Specifies the name of the role to select.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
select_role servicerep
```

This example retrieves the role definition named `servicerep` in the current zone and makes it as the currently selected role.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a role:

- `get_roles` returns a Tcl list of roles in the current zone.
- `list_roles` lists to `stdout` the roles in the current zone.

• • • • •

- `new_role` creates a new role and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with that role:

- `add_command_to_role` adds a UNIX command right to the current role.
- `add_pamapp_to_role` adds a PAM application right to the current role.
- `delete_role` deletes the selected role from Active Directory and from memory.
- `get_role_apps` returns a Tcl list of the PAM application rights associated with the current role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the current role.
- `get_role_field` reads a field value from the current role.
- `list_role_rights` returns a list of all UNIX command and PAM application rights associated with the current role.
- `remove_command_from_role` removes a UNIX command right from the current role.
- `remove_pamapp_from_role` removes a PAM application right from the current role.
- `save_role` saves the selected role with its current settings to Active Directory.
- `set_role_field` sets a field value in the current role.

## `select_role_assignment`

Use the `select_role_assignment` command to retrieve a role assignment in the currently selected zone from Active Directory. This command stores the role assignment in memory, and makes it the currently selected role assignment for subsequent ADEdit commands. The role assignment remains selected until you select another role assignment or zone, delete the role assignment, or end the ADEdit session.

If you use ADEdit commands such as `set_role_assignment_field` to change settings for the selected role assignment, you must save the selected role assignment using the `save_role_assignment` command for your changes to



take effect in Active Directory. If you select another role assignment or end the ADEdit session before saving the currently selected role assignment, your changes will be lost.

You can only use the `select_role_assignment` command to select role assignments if the currently selected zone is a classic4 or hierarchical zone. The command does not work for other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
select_role_assignment principal/role[/zone]
```

## Abbreviation

slra

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>principal/role</code> <code>[/zone]</code>	string	Required. Specifies the user principal name (UPN) of the user or group to whom the role is assigned, followed by a slash (/) and the name of the role to assign to the principal. The <i>zone</i> argument is optional if the role is defined in the currently selected zone. If the role is defined in a zone other than the currently selected zone, the <i>/zone</i> argument is required.

• • • • •

## Return value

This command returns nothing if it runs successfully.

## Examples

```
select_role_assignment poweradmins@acme.com/root/global
```

This example retrieves the role assignment that assigns the role named `root`, as defined in the `global` zone, to the principal named `poweradmins@acme.com`. The principal is a group.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a role assignment:

- `get_role_assignments` returns a Tcl list of role assignments in the current zone.
- `list_role_assignments` lists to `stdout` the role assignments in the current zone.
- `new_role_assignment` creates a new role assignment and stores it in memory.
- `select_role_assignment` retrieves a role assignment from Active Directory and stores it in memory.

After you have a role assignment stored in memory, you can use the following commands to work with that role assignment:

- `delete_role_assignment` deletes the selected role assignment from Active Directory and from memory.
- `get_role_assignment_field` reads a field value from the currently selected role assignment.
- `save_role_assignment` saves the selected role assignment with its current settings to Active Directory.



- `set_role_assignment_field` sets a field value in the currently selected role assignment.

## select\_rs\_command

Use the `select_rs_command` command to retrieve a restricted shell command in the currently selected zone from Active Directory, store it in memory, and set it as the currently selected restricted shell command for other ADEdit commands. After you select the restricted shell command to work with, it remains selected until you select a different restricted shell command, change the currently selected zone, delete the restricted shell command, or end the ADEdit session.

If you use ADEdit commands such as `set_rsc_field` to change settings for the selected restricted shell command, you must save the restricted shell command using the `save_rs_command` command for your changes to take effect in Active Directory. If you select another restricted shell command or end the ADEdit session before saving the currently selected restricted shell command, your changes will be lost.

You can only use the `select_rs_command` if the currently selected zone is a classic zone. The command does not work in other types of zones.

## Zone type

Classic only

## Syntax

```
select_rs_command rs_cmd
```

## Abbreviation

```
slrsc
```

.....

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
rs_cmd	string	Required. Specifies the name of the restricted shell command to select.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
select_rs_command rsc1
```

This command looks for the restricted shell command name `rsc1` in the current zone. If `rsc1` is found in the current zone, it becomes the currently selected context for subsequent commands.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the restricted shell command to work with:

- `get_rs_commands` returns a Tcl list of restricted shell commands in the current zone.
- `list_rs_commands` lists to `stdout` the restricted shell commands in the current zone.
- `new_rs_command` creates a new restricted shell command and stores it in memory.



After you have a restricted shell command stored in memory, you can use the following commands to work with that restricted shell command:

- `delete_rs_command` deletes the selected command from Active Directory and from memory.
- `get_rsc_field` reads a field value from the currently selected command.
- `save_rs_command` saves the selected command with its current settings to Active Directory.
- `set_rsc_field` sets a field value in the currently selected command.

## select\_rs\_env

Use the `select_rs_env` command to retrieve a restricted shell environment in the currently selected zone from Active Directory, stores it in memory, and sets it to be the currently selected restricted shell environment for other ADEdit commands. The restricted shell environment remains selected until you select another restricted shell environment, change the currently selected zone, delete the restricted shell environment, or end the ADEdit session.

If you use ADEdit commands such as `set_rse_field` to change settings for the restricted shell environment, you must save the restricted shell environment using the `save_rs_env` command for your changes to take effect in Active Directory. If you select another restricted shell environment or end the ADEdit session before saving the currently selected restricted shell environment, your changes will be lost.

You can only use the `select_rs_env` command if the currently selected zone is a classic4 zone. The command does not work in other types of zones.

## Zone type

Classic only

## Syntax

```
select_rs_env rse_name
```

.....

## Abbreviation

`slrse`

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>rse_name</code>	string	Required. Specifies the name of the restricted shell environment to select.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
select_rs_env rse1
```

This command looks for the restricted shell environment named `rse1` in the current zone. If `rse1` is found in the current zone, it becomes the currently selected context for subsequent commands.

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with restricted shell environments:

• • • • •

- `get_rs_envs` returns a Tcl list of restricted shell environments.
- `list_rs_envs` lists to `stdout` the restricted shell environments.
- `new_rs_env` creates a new restricted shell environment and stores it in memory.

After you have a restricted shell environment stored in memory, you can use the following commands to work with its fields:

- `delete_rs_env` deletes the current restricted shell environment from Active Directory and from memory.
- `get_rse_field` reads a field value from the current restricted shell environment.
- `save_rs_env` saves the restricted shell environment to Active Directory.
- `set_rse_field` sets a field value in the current restricted shell environment.

## select\_zone

Use the `select_zone` command to retrieve a zone from Active Directory, stores the zone in memory, and make that zone as the currently selected zone for subsequent ADEdit commands. The zone remains selected until you select another zone, delete the zone, or end the ADEdit session.

If you use ADEdit commands such as `set_zone_field` to change settings for the zone, you must save the zone using the `save_zone` command for your changes to take effect in Active Directory. If you select another zone or end the ADEdit session before saving the currently selected zone, your changes will be lost.

You should note that ADEdit treats *computer roles* and *computer-specific overrides* as special use-case zones. You can, therefore, use the `select_zone` command to retrieve a “computer role zone” or a “computer-specific zone” to work with as the currently selected zone. If you specify a zone that is a computer role zone or a computer-specific zone, subsequent ADEdit commands will treat the zone as a computer role or a computer-specific zone instead of a standard zone. You can only work with one zone at a time, regardless of type. Because some ADEdit commands behave differently in



different types of zones, you should verify the type of zone you are working with when you select a zone.

## Zone type

Classic and hierarchical

## Syntax

```
select_zone [-nc] path
```

## Abbreviation

slz

## Options

This command takes the following option:

Option	Description
	Requests a reread of the zone's fields from Active Directory.
-nc	Use this option after you use the <code>save_zone</code> command to ensure you have the current Active Directory field values in memory. For example, after a <code>save_zone</code> command, the <code>modifyTime</code> field value is updated. If you do not then run <code>select_zone -nc</code> , a <code>gzf modifyTime</code> command returns the previous value.

## Arguments

This command takes the following argument:

Argument	Type	Description
		Required. Specifies the path to the selected zone or computer role. The path format depends on the type of zone selected: <ul style="list-style-type: none"> <li>■ A tree, classic3, classic4, or SFU zone path consists of the zone's distinguished name. Enclose the path in braces or quotes to allow spaces in the distinguished name.</li> </ul>
path	string	<ul style="list-style-type: none"> <li>■ A computer role path consists of the host zone's distinguished name followed by a slash (/) and the name of the computer zone. Enclose the path in braces or quotes to allow spaces in the distinguished name.</li> <li>■ A computer override path consists of the computer name followed by an ampersand (@) and the distinguished name of the host zone.</li> </ul>

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example selects a standard zone named `cz1` in the `Zones` container in the `UNIX` organizational unit in the `acme.com` domain:

```
select_zone "CN=cz1,CN=Zones,OU=UNIX,DC=acme,DC=com"
```

The following example selects the computer role named `LinuxComputers` in the `global` zone in the `Zones` container in the `UNIX` organizational unit in the `acme.com` domain:

```
select_zone "CN=global,CN=Zones,OU=UNIX,DC=acme,DC=com/LinuxComputers"
```

The following example selects the computer-specific override zone named `server1` in the `global` zone in the `acme.com` domain:

```
select_zone server1@"CN=global,CN=Zones,OU=Centrify,DC=acme,DC=com"
```



## Related commands

The following commands perform actions related to this command:

- `create_zone` creates a new zone in Active Directory.
- `get_zones` returns a Tcl list of all zones within a specified domain.

After you have a zone stored in memory, you can use the following commands to work with that zone:

- `delegate_zone_right` delegates a zone use right to a specified user or computer.
- `delete_zone` deletes the selected zone from Active Directory and memory.
- `get_child_zones` returns a Tcl list of child zones, computer roles, or computer zones.
- `get_zone_field` reads a field value from the currently selected zone.
- `get_zone_nss_vars` returns the NSS substitution variable for the selected zone.
- `save_zone` saves the selected zone with its current settings to Active Directory.
- `set_zone_field` sets a field value in the currently selected zone.

## select\_zone\_computer

Use the `select_zone_computer` command to retrieve a zone computer in the currently selected zone from Active Directory, store it in memory, and make it the currently selected zone computer for subsequent ADEdit commands. The zone computer remains selected until you select another zone computer, delete the zone computer, or end the ADEdit session.

If you use ADEdit commands such as `set_zone_computer_field` to change settings for the zone computer, you must save the zone computer using the `save_zone_computer` command for your changes to take effect in Active Directory. If you select another zone computer or end the ADEdit session before saving the currently selected zone computer, your changes will be lost.



## Zone type

Classic and hierarchical

## Syntax

```
select_zone_computer sAMAccountName$@domain
```

## Abbreviation

slzc

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
		Required. Specifies the Active Directory computer's <code>sAMAccountName</code> followed by <code>\$@</code> and the computer's domain.
<code>sAMAccountName</code>	string	You can look up the <code>sAMAccountName</code> for a computer in Active Directory Users and Computers or by running the <code>get_zone_computers</code> command.

## Return value

This command returns nothing if it runs successfully.



## Examples

```
select_zone_computer sales2$@acme.com
```

This example looks for the zone computer named `sales2` in the current zone and, if found, selects it as the current zone computer.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and manage the zone computers:

- `get_zone_computers` returns a Tcl list of the Active Directory names of all zone computers in the current zone.
- `list_zone_computers` lists to `stdout` the zone computers in the current zone.
- `new_zone_computer` creates a new zone computer and stores it in memory.

After you have a zone computer stored in memory, you can use the following commands to work with that zone computer:

- `delete_zone_computer` deletes the zone computer from Active Directory and from memory.
- `get_zone_computer_field` reads a field value from the currently selected zone computer.
- `save_zone_computer` saves the zone computer with its current settings to Active Directory.
- `set_zone_computer_field` sets a field value in the currently selected zone computer.

## select\_zone\_group

Use the `select_zone_group` command to retrieve a zone group in the currently selected zone from Active Directory. The command stores the zone group in memory and makes it the currently selected zone group for



subsequent ADEdit commands. The zone group remains selected until you select another zone group, delete the zone group, or end the ADEdit session.

If you use ADEdit commands such as `set_zone_group_field` to change settings for the zone group, you must save the zone group using the `save_zone_group` command for your changes to take effect in Active Directory. If you select another zone group or end the ADEdit session before saving the currently selected zone group, your changes will be lost.

## Zone type

Classic and hierarchical

## Syntax

```
select_zone_group AD_group_UPN
```

## Abbreviation

slzg

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
AD_group_UPN	string	Required. Specifies the user principal name (UPN) of a zone group in the currently selected zone.



## Return value

This command returns nothing if it runs successfully.

## Examples

```
select_zone_group poweradmins@acme.com
```

This example looks for the group named `poweradmins` in the current zone and, if found, selects it as the current zone group.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select zone groups:

- `get_zone_groups` returns a Tcl list of the Active Directory names of all zone groups in the current zone.
- `list_zone_groups` lists to `stdout` the zone groups in the current zone.
- `new_zone_group` creates a new zone group and stores it in memory.

After you have a zone group stored in memory, you can use the following commands to work with that zone group:

- `delete_zone_group` deletes the selected zone group from Active Directory and from memory.
- `get_zone_group_field` reads a field value from the currently selected zone group.
- `save_zone_group` saves the selected zone group with its current settings to Active Directory.
- `set_zone_group_field` sets a field value in the currently selected zone group.



## select\_zone\_user

Use the `select_zone_user` command to retrieve a zone user in the currently selected zone from Active Directory. This command stores the zone user in memory, and makes it the currently selected zone user for subsequent ADEdit commands. The zone user remains selected until you select another zone user, delete the zone user, or end the ADEdit session.

If you use ADEdit commands such as `set_zone_user_field` to change settings for the zone user, you must save the zone user using the `save_zone_user` command for your changes to take effect in Active Directory. If you select another zone user or end the ADEdit session before saving the currently selected zone user, your changes will be lost.

## Zone type

Classic and hierarchical

## Syntax

```
select_zone_user user
```

## Abbreviation

```
slzu
```

## Options

This command takes no options.

## Arguments

This command takes the following argument:



---

Argument	Type	Description
		Required. Specifies the sAMAccountName@domain or user principal name (UPN) of a zone user in the currently selected zone.
user	string	ADEdit resolves the user with the sAMAccountName first, then the UPN. If the zone user is an orphan user—that is, the corresponding Active Directory user no longer exists—you must specify the user’s security identifier (SID) instead.

---

## Return value

This command returns nothing if it runs successfully.

## Examples

```
select_zone_user adam.avery@acme.com
```

This example looks for the Active Directory user `adam.avery` in the current zone and, if found, selects that user as the current zone user.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone user:

- `get_zone_users` returns a Tcl list of the Active Directory names of all zone users in the current zone.
- `list_zone_users` lists to `stdout` the zone users and their NSS data in the current zone.
- `new_zone_user` creates a new zone user and stores it in memory.
- `select_zone_user` retrieves a zone user from Active Directory and stores it in memory.

After you have a zone user stored in memory, you can use the following commands to work with that zone user:



- `delete_zone_user` deletes the selected zone user from Active Directory and from memory.
- `get_zone_user_field` reads a field value from the currently selected zone user.
- `save_zone_user` saves the selected zone user with its current settings to Active Directory.
- `set_zone_user_field` sets a field value in the currently selected zone user.

## set\_dzc\_field

Use the `set_dzc_field` command to set the value for a specified field in the currently selected UNIX command stored in memory. The `set_dzc_field` command does *not* set a field value stored in Active Directory for the selected UNIX command.

If you change any fields, you must save the UNIX command using the `save_dz_command` command for your changes to take effect in Active Directory. If you select another UNIX command or end the AEdit session before saving the currently selected UNIX command, your changes will be lost.

You can only use the `set_dzc_field` command to set UNIX command fields if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

When executing privileged commands on computers running Security-Enhanced Linux (SELinux), the security context contains additional information that is used to make access control decisions.

## Zone type

Classic and hierarchical

## Syntax

```
set_dzc_field field value
```

• • • • •

## Abbreviation

`sdzcf`

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
		Required. Specifies the name of the field you want to set. The possible values are:
		<ul style="list-style-type: none"> <li>■ <b>description:</b> Text describing the UNIX command.</li> <li>■ <b>cmd:</b> The UNIX command string or strings. You can use wild cards or a regular expression.</li> <li>■ <b>path:</b> The path to the command's location. You can use wild cards or a regular expression.</li> <li>■ <b>form:</b> An integer that indicates whether the cmd and path strings use wild cards (0) or a regular expression (1).</li> <li>■ <b>dzdo_runas:</b> A list of users and groups that can run this command under dzdo (similar to sudo). Users can be listed by user name or UID.</li> </ul>
field	string	<ul style="list-style-type: none"> <li>■ <b>dzsh_runas:</b> A list of users and groups that can run this command in a restricted shell environment (dzsh). Users can be listed by user name or UID. You cannot set this field value if the selected zone is a classic4 zone.</li> <li>■ <b>keep:</b> A comma-separated list of environment variables from the current user's environment to keep.</li> <li>■ <b>del:</b> A comma-separated list of environment variables from the current user's environment to delete.</li> <li>■ <b>add:</b> A comma-separated list of environment variables to add to the final set of environment variables.</li> <li>■ <b>pri:</b> An integer that specifies the command priority for the restricted shell command object.</li> </ul>
field (continued)	string	<ul style="list-style-type: none"> <li>■ <b>umask:</b> An integer that defines who can execute the command.</li> <li>■ <b>flags:</b> An integer that specifies a combination of different properties for the command.</li> <li>■ <b>selinux_role:</b> Specifies the SELinux role to use when</li> </ul>



Argument	Type	Description
		constructing a new security context for command execution. <ul style="list-style-type: none"><li>▪ <b>selinux_type</b>: Specifies the SELinux type to use when constructing a new security context for command execution.</li><li>▪ <b>digest</b>: Specifies the SHA-2 digest to verify the file checksum before command execution.</li><li>▪ Note that <code>selinux_role</code> and <code>selinux_type</code> are only supported on Red Hat Enterprise Linux systems and effective only on systems with SELinux enabled and joined to a hierarchical zone.</li></ul>
value		Required. Specifies the value to assign to the specified field. The data type depends on the field specified.  In most cases, you can assign an empty string or null value (0) to unset a field value, depending on the data type of the field.

## Setting the cmd and path field values

You can specify the `cmd` and `path` strings using wild cards (\*, ?, and !), or as a regular expression. If you specify the `cmd` and `path` strings using wild cards, use an asterisk (\*) to match zero or more characters, the question mark (?) to match exactly one character, or the exclamation mark (!) to negate matching of the specified string.

To set to the command path to the equivalent of the **Standard user path** option, you can set the value of the `path` field to `USERPATH`. To set to the path to the equivalent of the **Standard system path** option, set the value of the `path` field to `SYSTEMPATH`. To set to the path to the equivalent of the **System search path** option, set the value of the `path` field to `SYSTEMSEARCHPATH`.

For both the `cmd` and `path` fields, the `form` field controls whether the specified string is interpreted as a regular expression or as a string that includes wild cards.



## Specifying the environment variables to use

You can use the `keep`, `del`, and `add` settings to control the environment variables used by the commands specified by the `cmd` string. The `keep` and `del` settings are mutually exclusive. The `keep` field only takes effect if the `flag 16` is included in the setting for the `flag field`. The `del` field only takes effect if the `flag 16` is not included in the setting for the `flag field`.

Any environment variables kept or deleted are in addition to the default set of the user's environment variables that are either retained or deleted. The default set of environment variables to keep is defined in the `dzdo.env_keep` configuration parameter in the `centrifdc.conf` file. The default set of environment variables to delete is defined in the `dzdo.env_delete` configuration parameter in the `centrifdc.conf` file. You can also add environment variables to the final set of environment variables resulting from the `keep` or `del` fields.

## Specifying the command priority

You can use the `pri` field to specify the command priority when there are multiple matches for the UNIX commands specified by wild cards. If commands specified by this UNIX command object match commands specified by another UNIX command object, the UNIX command object with the higher command priority prevails. This field takes an integer value; the higher the number, the higher the priority.

## Specifying the umask value

You can use the `umask` field to define who can execute the command. The `umask` field specifies a 3-digit octal value that defines read, write, or execute permission for owner, group, and other users. The left digit defines the owner execution rights, the middle digit defines the group execution rights, and the right digit defines other execution rights. Each digit is a combination of binary flags, one flag for each right as follows:

- 4 is read
- 2 is write

• • • • •

- 1 is execute

You add these values add together to define the rights available for each entity. For example, an `umask` value of 600 indicates read and write permission (4+2) for the owner, but no permissions for the group or other users.

Similarly, an `umask` value of 740 indicates read, write, execute permissions (4+2+1) for the owner, read permissions for the group, but no permissions for other users.

## Specifying command properties using the `flags` field

You can use the `flags` field to define a combination of binary flags, with one flag for each of the following properties:

- 1**—Prevents nested command execution. If this flag value is not set, nested command execution is allowed.
- 2**—Requires re-authentication using the login user's password.
- 4**—Requires authentication using the run-as user's password.
- 8**—Preserves group membership. If this flag value is not set, group membership is not preserved.
- 16**—Resets environment variables for the command, deleting the variables specified in the `dzdo.env_delete` parameter and keeping the variables specified in the `keep` field. If this flag is not set, the command removes the unsafe environment variables specified in the `dzdo.env_delete` parameter along with any additional environment variables specified by the `del` field.
- 32**—Requires multi-factor authentication to execute the command.
- 64**—Prevents navigation up the path hierarchy when executing the command.

You add these values together to define the setting for the `flags` field. For example, a `flags` field value of 5 prevents nested command execution and requires authentication using the run-as user's password (1+4). You cannot set the 2 flag and the 4 flag or the 4 flag and the 32 flag simultaneously. If you don't set any of these flags, re-authentication is not required.



## Return value

This command returns nothing if it runs successfully.

## Examples

The following example sets the current UNIX command `dzdo_runas` field to `root`:

```
set_dzc_field dzdo_runas root
```

The following example sets the UNIX command properties so that nested command execution is not allowed and authentication is required with the user's password:

```
sdzcf flags 3
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a UNIX command to work with:

- [get\\_dz\\_commands](#) returns a Tcl list of UNIX commands in the current zone.
- [list\\_dz\\_commands](#) lists to `stdout` the UNIX commands in the current zone.
- [new\\_dz\\_command](#) creates a new UNIX command and stores it in memory.
- [select\\_dz\\_command](#) retrieves a UNIX command from Active Directory and stores it in memory.

After you have a UNIX command stored in memory, you can use the following commands to work with that command:

- [delete\\_dz\\_command](#) deletes the selected command from Active Directory and from memory.
- [get\\_dzc\\_field](#) reads a field value from the currently selected command.



- `save_dz_command` saves the selected command with its current settings to Active Directory.

## set\_ldap\_timeout

Use the `set_ldap_timeout` command to set the time-out interval used by LDAP commands. LDAP commands are ADEdit commands such as `select_zone` that perform read/write operations on Active Directory through a binding. The time-out value controls how long these commands will wait for a response before declaring a time-out and ceasing operation.

The default value is five minutes.

### Zone type

Not applicable

### Syntax

```
set_ldap_timeout timeout_in_seconds
```

### Abbreviation

None.

### Options

This command takes no options.

### Arguments

This command takes the following argument:



Argument	Type	Description
timeout_in_seconds	integer	Required. Specifies the number of seconds to wait for a response from Active Directory before ending an operation. The default value is 300 seconds (5 minutes).

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_ldap_timeout 120
```

This example sets the LDAP time-out interval to 120 seconds (2 minutes).

## Related commands

None.

## set\_local\_group\_profile\_field

Use the `set_local_group_profile_field` command to set the value of the specified profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone. Before executing this command, you must create a new local group by executing the `new_local_group_profile` command, or select an existing local group by executing the `select_local_group_profile` command.

You can save a group object before the group profile is complete. However, the group profile is not added to `/etc/group` on each UNIX and Linux computer in the zone until the group profile is complete and the `profileflag` field is set to 1 (enabled). See [new\\_local\\_group\\_profile](#) for details about which fields (attributes) a group profile must have to be considered complete.

.....

## Zone type

Hierarchical only.

## Syntax

```
set_local_group_profile_field field_name value
```

## Abbreviation

slgpf

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
field_name	string	Required. Specifies the local group profile field to set. The possible values are: <ul style="list-style-type: none"><li>▪ <b>gid</b></li><li>▪ <b>member</b></li><li>▪ <b>profileflag</b></li></ul> You can also specify AIX extended attributes as the field to set an extended attribute value for a group. Extended attribute fields start with the <code>aix.</code> prefix. For example, the <code>admin</code> extended attribute can be set by specifying <code>aix.admin</code> as the field.
value		Required. The data type depends on the field being set. The possible values for each field are: <ul style="list-style-type: none"><li>▪ <b>Any field:</b> Clear any field by entering a hyphen</li></ul>

Argument	Type	Description
		character (-).
		<ul style="list-style-type: none"> <li>▪ <b>gid:</b> A numeric group identifier.</li> <li>▪ <b>member:</b> The UNIX name of a local user to add to the group.</li> <li>▪ <b>profileflag:</b> 1 or 3.            If set to 1, the group profile is enabled. If the group profile is complete and the profile flag is set to 1, the profile will be installed or updated in <code>/etc/group</code> at the next local account refresh interval.             If set to 3, the group profile is removed from <code>/etc/group</code> at the next local account refresh interval.</li> </ul>

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example sets the GID of the currently selected group to 20001.

```
set_local_group_profile_field gid 20001
```

The following example adds the UNIX user `anton.splieth` to the currently selected local group.

```
set_local_group_profile_field member anton.splieth
```

The following example sets the profile flag of the currently selected group to 1 (enabled), so that if the group profile is complete, the profile will be installed or updated in `/etc/group` at the next local account refresh interval.

```
set_local_group_profile_field profileflag 1
```

If the current group is on AIX, you can set group AIX extended attributes and values. For example, to identify the current group as an administrative group, you can set the `admin` extended attribute:

```
set_local_group_profile_field aix.admin true
```



## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- `delete_local_group_profile` deletes a local UNIX or Linux group that has a profile defined in the current zone.
- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.
- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.



- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_user_profile_field` sets the value of a field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.

## set\_local\_user\_profile\_field

Use the `set_local_user_profile_field` command to set the value of the specified profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone. Before executing this command, you must create a new local user by executing the `new_local_user_profile` command, or select an existing local user by executing the `select_local_user_profile` command.

You can save a user object before the user profile is complete. However, the user profile is not added to `/etc/passwd` on each UNIX and Linux computer in the zone until the user profile is complete, the `profileflag` field is set to 1 (enabled) or 2 (disabled), and the user is assigned a visible role such as `local` listed. See `new_local_user_profile` for details about which attributes a user profile must have to be considered complete.

## Zone type

Hierarchical only.

## Syntax

```
set_local_user_profile_field field_name value
```

## Abbreviation

slupf

.....

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
field_name	String	<p>Required. Specifies the local user profile field to set. Fields and possible values are:</p> <ul style="list-style-type: none"> <li>▪ <b>Any field:</b> Clear any field by entering a hyphen (-).</li> <li>▪ <b>uid:</b> The user’s numeric identifier.</li> <li>▪ <b>gid:</b> The user’s primary group numeric identifier.</li> <li>▪ <b>shell:</b> The local user’s default shell on the local computer. Possible values are: /bin/bash, /bin/csh, /bin/ksh, /bin/sh, /bin/tcsh, %<code>{shell}</code>.</li> <li>▪ <b>home:</b> The local user’s default home directory on the local computer.</li> <li>▪ <b>gecos:</b> General information about the local user account.</li> <li>▪ <b>profileflag:</b> The value of the user’s profile flag as set in the user object in the zone. For the user to be managed by the agent, the profile flag must be set to 1, 2, or 3. <ul style="list-style-type: none"> <li>If set to 1, the user profile is enabled. If the user profile is complete, the profile flag is set to 1, and the user is assigned a visible role, the profile will be installed or updated in <code>/etc/passwd</code> at the next local account refresh interval.</li> <li>If set to 2, the user profile is disabled. If the user profile is complete, the profile flag is set to 2, and the user is assigned a visible role, the profile will be installed or updated in <code>/etc/passwd</code> at the next local account refresh interval. However, the password field in <code>/etc/passwd</code> will be set to <code>!!</code>, and the user will not be able to log into the local computer. This state results in what is typically called a “locked account.”</li> <li>If set to 3, the user profile is removed from <code>/etc/passwd</code> at the next local account refresh interval.</li> </ul> </li> </ul>

## Return value

This command returns nothing if it runs successfully.

• • • • •

## Examples

The following example sets the UID of the currently selected user to 10001.

```
set_local_user_profile_field uid 10001
```

The following example sets the primary group ID for the currently selected user to 20001.

```
set_local_user_profile_field gid 20001
```

The following example sets the default shell for the currently selected user to `/bin/csh`:

```
set_local_user_profile_field shell /bin/csh
```

The following example sets the home directory for the currently selected user to `/home`.

```
set_local_user_profile_field home /home
```

The following example sets the profile flag of the currently selected user to 1 (enabled), so that if the user profile is complete and the user is assigned a visible role, the profile will be installed or updated in `/etc/passwd` at the next local account refresh interval.

```
set_local_user_profile_field profileflag 1
```

## Related commands

The following related AEdit commands let you view and administer local UNIX and Linux users and groups that have profiles defined in the current zone:

- `delete_local_group_profile` deletes a local UNIX or Linux group that has a profile defined in the current zone.
- `delete_local_user_profile` deletes a local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_group_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.



- `get_local_groups_profile` displays a TCL list of profiles for local groups that are defined in the current zone.
- `get_local_user_profile_field` displays the value of a profile field for the currently selected local UNIX or Linux user that has a profile defined in the current zone.
- `get_local_users_profile` displays a TCL list of profiles for local users that are defined in the current zone.
- `list_local_groups_profile` displays a list of local UNIX and Linux groups that have a profile defined in the current zone.
- `list_local_users_profile` displays a list of local UNIX and Linux users that have a profile defined in the current zone.
- `new_local_group_profile` creates an object for a local UNIX or Linux group in the currently selected zone.
- `new_local_user_profile` creates an object for a local UNIX or Linux user in the currently selected zone.
- `save_local_group_profile` saves the currently selected local UNIX or Linux group object after you create the group object or edit profile field values in the group object.
- `save_local_user_profile` saves the currently selected local UNIX or Linux user object after you create the user object or edit profile field values in the user object.
- `select_local_group_profile` selects a local UNIX or Linux group object for viewing or editing.
- `select_local_user_profile` selects a local UNIX or Linux user object for viewing or editing.
- `set_local_group_profile_field` sets the value of a field for the currently selected local UNIX or Linux group that has a profile defined in the current zone.

## set\_object\_field

Use the `set_object_field` command to set the value for a specified field in the currently selected Active Directory object stored in memory. The `set_`



`object_field` command does *not* set a field value stored in Active Directory for this object.

If you change any fields, you must save the object using the `save_object` command for your changes to take effect in Active Directory. If you select another object or end the ADEdit session before saving the currently selected object, your changes will be lost.

The `set_object_field` command does not check to see if fields and values are valid. When you save an object, Active Directory will check fields and values at that time and report an error if they aren't valid.

## Zone type

Not applicable

## Syntax

```
set_object_field field value
```

## Abbreviation

sof

## Options

This command takes no options.

## Arguments

This command takes the following arguments:



Argument	Type	Description
field	string	Required. Specifies the name of the field you want to set. The <i>field</i> argument can be any attribute that is valid for the type of Active Directory object currently selected in memory.
value		Required. Specifies the value to assign to the specified field. The data type depends on the specified field. The <code>set_object_field</code> command does not check whether the value is valid. Active Directory will check for valid values when ADEdit saves the object.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_object_field sd $sdvalue
```

This example sets the current object's security descriptor field to the string contained in the variable `sdvalue` (an SDDL string).

## Related commands

The following commands enable you to view and select Active Directory objects:

- [get\\_objects](#) performs an LDAP search of Active Directory and returns a Tcl list of the distinguished names of objects matching the specified search criteria.
- [new\\_object](#) creates a new Active Directory object and stores it in memory.
- [select\\_object](#) retrieves an object with its attributes from Active Directory and stores it in memory.

After you have an object stored in memory, you can use the following commands to work with that object:



- `add_object_value` adds a value to a multi-valued field attribute of the currently selected Active Directory object.
- `delete_object` deletes the selected Active Directory object from Active Directory and from memory.
- `delete_sub_tree` deletes an Active Directory object and all of its children from Active Directory.
- `get_object_field` reads a field value from the currently selected Active Directory object.
- `remove_object_value` removes a value from a multi-valued field attribute of the currently selected Active Directory object.
- `save_object` saves the selected Active Directory object with its current settings to Active Directory.

## set\_pam\_field

Use the `set_pam_field` command to set the value for a specified field in the currently selected PAM application right stored in memory. The `set_pam_field` command does *not* set a field value stored in Active Directory for this PAM application right.

If you change any fields, you must save the PAM application right using the `save_pam_app` command for your changes to take effect in Active Directory. If you select another PAM application right or end the ADEdit session before saving the currently selected PAM application right, your changes will be lost.

You can only use the `set_pam_field` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
set_pam_field field value
```

## Abbreviation

spf

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
		<p>Required. Specifies the name of the field that you want to set. The possible values are:</p> <ul style="list-style-type: none"> <li>▪ <b>application:</b> The name of the PAM application that is allowed to use the <code>adcli</code> PAM authentication service. The name can be literal, or it can contain <code>?</code> or <code>*</code> wildcard characters to specify multiple applications.</li> <li>▪ <b>description:</b> Text describing the PAM application.</li> </ul> <p>Note that in a classic zone, setting the <b>application</b> field changes the name of the PAM application right. For example, assume you create a new PAM application right in a classic zone using a command like this:</p> <pre>new_pam_app myftp</pre> <p>If you then use this command to set the <b>application</b> field like this:</p> <pre>set_pam_field application newftp</pre> <p>The PAM application right itself will be renamed. If you were to use the <code>list_pam_apps</code> command after running the <code>set_pam_field</code> command, the right would be returned as <code>newftp</code>:</p> <pre>list_pam_apps newftp : Renamed application right</pre>
field	string	
value		<p>Required. Specifies the value to assign to the specified field.</p> <p>In most cases, you can assign an empty string to unset a field value.</p>

• • • • •

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_pam_field application *
```

This example sets the `application` field for the current PAM application right to allow PAM access rights to all applications (\* is the wildcard for all possible strings).

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select PAM application rights:

- `get_pam_apps` returns a Tcl list of PAM application rights in the current zone.
- `list_pam_apps` lists to `stdout` the PAM application rights in the currently selected zone.
- `new_pam_app` creates a new PAM application right and stores it in memory.
- `select_pam_app` retrieves a PAM application right from Active Directory and stores it in memory.

After you have a PAM application right stored in memory, you can use the following commands to work with that PAM application right:

- `delete_pam_app` deletes the selected PAM application right from Active Directory and from memory.
- `get_pam_field` reads a field value from the currently selected PAM application right.
- `save_pam_app` saves the selected PAM application right with its current settings to Active Directory.

.....

## set\_role\_assignment\_field

Use the `set_role_assignment_field` command to sets the value for a specified field in the currently selected role assignment stored in memory. The `set_role_assignment_field` command does *not* set a field value stored in Active Directory for this role assignment.

If you change any fields, you must save the role assignment using the `save_role_assignment` command for your changes to take effect in Active Directory. If you select another role assignment or end the ADEdit session before saving the currently selected role assignment, your changes will be lost.

You can only use the `set_role_assignment_field` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
set_role_assignment_field field value
```

## Abbreviation

sraf

## Options

This command takes no options.

## Arguments

This command takes the following arguments:



Argument	Type	Description
field	string	<p>Required. Specifies the name of the field that you want to set. The possible values are:</p> <ul style="list-style-type: none"><li>▪ <b>customAttr</b>: Sets custom text strings for the role assignment. This field is only applicable for hierarchical zones.</li><li>▪ <b>description</b>: Sets the description for the role assignment.</li><li>▪ <b>from</b>: Sets the starting date and time for the role assignment. The date and time is expressed in standard UNIX time. The Tcl <code>clock</code> command manipulates these time values. A value of 0 means no starting date and time for the role assignment.</li><li>▪ <b>role</b>: Sets the name of the role to assign and the zone in which the role was defined. The zone value is optional if the role is defined in the currently selected zone. The zone is required if the role is defined in another zone.</li><li>▪ <b>to</b>: Sets the ending date and time for the role assignment. The start and end dates and times are expressed in standard UNIX time. You can use the Tcl <code>clock</code> command to manipulate these values. A value of 0 indicates no date or time is set for the role assignment.</li></ul>
value	depends on field	<p>Required. Specifies the value to assign to the specified field.</p> <p>In some cases, you can assign a dash (-) or an empty string to unset a field value. However, this is not supported for all fields or all zone types.</p>

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_role_assignment_field role su-root/global
```



This example assigns the role named `su-root` that is defined in the `global` zone.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a role assignment:

- `get_role_assignments` returns a Tcl list of role assignments in the current zone.
- `list_role_assignments` lists to `stdout` the role assignments in the current zone.
- `new_role_assignment` creates a new role assignment and stores it in memory.
- `select_role_assignment` retrieves a role assignment from Active Directory and stores it in memory.

After you have a role assignment stored in memory, you can use the following commands to work with that role assignment:

- `delete_role_assignment` deletes the selected role assignment from Active Directory and from memory.
- `get_role_assignment_field` reads a field value from the currently selected role assignment.
- `save_role_assignment` saves the selected role assignment with its current settings to Active Directory.

## set\_role\_field

Use the `set_role_field` command to set the value for a specified field in the currently selected role stored in memory. The `set_role_field` does *not* set a field value stored in Active Directory for this role.

If you change any fields, you must save the role using the `save_role` command for your changes to take effect in Active Directory. If you select another role or



end the ADEdit session before saving the currently selected role, your changes will be lost.

You can only use the `set_role_field` command if the currently selected zone is a classic4 or hierarchical zone. The command does not work in other types of zones.

## Zone type

Classic and hierarchical

## Syntax

```
set_role_field field value
```

## Abbreviation

srf

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
field	string	Required. Specifies the name of the field that you want to set.
		Required. Specifies the value to assign to the specified field.
value		In most cases, you can assign an empty string or null value (0) to unset a field value, depending on the data type of the field.



The data type required depends on the field you are setting. The possible values are:

- **allowLocalUser:** Set the value to true to allow local users to be assigned to the role, or false if local users should not be assigned to the role. This field is not applicable in classic zones. The valid values are 1, y, yes, or true to enable or 0, n, no, or false to disable. All other values throw an exception.
- **AlwaysPermitLogin:** Set the value to true to enable “rescue rights” for users assigned to the role, or false if “rescue rights” should not be applied to the role. This field is not applicable in classic zones. The valid values are 1, y, yes, or true to enable or 0, n, no, or false to disable. All other values throw an exception.
- **auditLevel:** Set the value to one of the following to specify whether auditing is not requested, requested but not required, or required:
  - AuditIfPossible
  - AuditNotRequested
  - AuditRequired

This field is not applicable in classic zones.
- **customAttr:** Sets custom text strings for the role. This field is only applicable for hierarchical zones.
- **description:** Set the value to a text string that describes the role.
- **sysrights:** Set the value to specify the system rights granted to the role. This value is an integer that represents a combination of binary flags, one for each right. This field is not applicable in classic zones.
- **timebox:** Set the value to indicate the hours in the week when the role is enabled. This value is a 42-digit hexadecimal number. When represented in binary, each bit represents an hour of the week as described in the appendix [Timebox value format](#)
- **visible:** Returns true or false depending on whether “User is visible” right is configured for the role. You cannot get this field value if the selected zone is a classic zone.



## Setting the system rights field value for a role

You can specify the `sysrights` field to define the system rights that you want to grant to the currently selected role. This field value is an integer that represents a combination of binary flags, with one flag for each of the following system rights:

- 1**—Password login and non password (SSO) login are allowed.
- 2**—Non password (SSO) login is allowed.
- 4**—Account disabled in Active Directory can be used by sudo, cron, etc.
- 8**—Log in with non-restricted shell.
- 16**—Audit not requested/required.
- 32**—Audit required.
- 64**—Always permit to login.
- 128**—Remote login access is allowed for Windows computers.
- 256**—Console login access is allowed for Windows computers.
- 512**—Require multi-factor authentication through the Centrify connector to log on.
- 1024**—PowerShell remote access is allowed

These values are added together to define the `sysrights` field value. For example, a `sysrights` value of 6 indicates that the role is configured to allow single sign-on login and to ignore disabled accounts (2+4). A value of 11 indicates that most common UNIX system rights are enabled (1+2+8). A value of 384 indicates that most common Windows system rights are enabled (128+256).

## Return value

This command returns nothing if it runs successfully.



## Examples

The following example sets the system rights for the current role to allow SSO login (2) and to provide a full shell (8):

```
set_role_field sysrights 10
```

The following example sets the current role to require auditing:

```
set_role_field auditLevel AuditRequired
```

Note that the `sysrights` field is a bit field, so you can add and remove bits for the field instead of setting the integer value directly. For example to add the system rights for single sign-on and full shell to existing system rights, you might use commands similar to this:

```
set sr [get_role_field sysrights]
set_role_field sysrights [expr { $sr | 10 }]
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select roles:

- [get\\_roles](#) returns a Tcl list of roles in the current zone.
- [list\\_roles](#) lists to stdout the roles in the current zone.
- [new\\_role](#) creates a new role and stores it in memory.
- [select\\_role](#) retrieves a role from Active Directory and stores it in memory.

After you have a role stored in memory, you can use the following commands to work with that role:

- [add\\_command\\_to\\_role](#) adds a UNIX command to the current role.
- [add\\_pamapp\\_to\\_role](#) adds a PAM application right to the current role.
- [delete\\_role](#) deletes the selected role from Active Directory and from memory.



- `get_role_apps` returns a Tcl list of the PAM applications associated with the currently selected role.
- `get_role_commands` returns a Tcl list of the UNIX commands associated with the current role.
- `get_role_field` reads a field value from the currently selected role.
- `list_role_rights` returns a list of all UNIX commands and PAM application rights associated with the current role.
- `remove_command_from_role` removes a UNIX command from the current role.
- `remove_pamapp_from_role` removes a PAM application from the current role.
- `save_role` saves the selected role with its current settings to Active Directory.

## set\_rs\_env\_for\_role

Use the `set_rs_env_for_role` command to assign a restricted shell environment to the currently selected role that is stored in memory. You should note that a role can only have one restricted shell environment assigned to it. If you assign a new restricted shell environment to a role, the current restricted shell environment—if one exists—will be removed. In addition, a role cannot be defined with both privileged commands and a restricted shell environment at the same time. If you assign a restricted shell environment to the currently selected role, all privileged commands previously defined for the role—if they exist—will be removed from the role.

The `set_rs_env_for_role` command does not modify the data stored in Active Directory for the restricted shell environment. If you run this command using ADEdit without saving the role to Active Directory, your changes do not take effect.

You can only use the `set_rs_env_for_role` command if the currently selected zone is a classic4 zone. The command does not work in other types of zones.

.....

## Zone type

Classic only

## Syntax

```
set_rs_env_for_role environment
```

## Abbreviation

```
srse
```

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
environment	string	Required. Specifies the name of the restricted shell environment to assign to the current role.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_rs_env_for_role rsel
```



This example sets the currently selected role's restricted shell environment to `rse1`, and removes any existing restricted shell environment or privileged commands if they exist in the role.

## Related commands

The following commands perform actions related to this command:

- `clear_rs_env_from_role` removes a restricted shell environment from the current role.
- `get_rs_envs` returns a Tcl list of restricted shell environments.
- `list_rs_envs` lists to `stdout` the restricted shell environments.
- `new_rs_env` creates a new restricted shell environment and stores it in memory.
- `select_rs_env` retrieves a restricted shell environment from Active Directory and stores it in memory.

After you have a restricted shell environment stored in memory, you can use the following commands to work with that: restricted shell environment:

- `delete_rs_env` deletes the current restricted shell environment from Active Directory and from memory.
- `get_rse_field` reads a field value from the current restricted shell environment.
- `save_rs_env` saves the restricted shell environment to Active Directory.

## set\_rsc\_field

Use the `set_rsc_field` command to set the value for a specified field for the currently selected restricted shell command that is stored in memory. The `set_rsc_field` command does not set the field value stored in Active Directory for the selected restricted command field.

If you change any fields, you must save the restricted shell command using the `save_rs_command` command for your changes to take effect in Active Directory. If you select another restricted shell command or end the ADEdit



session before saving the currently selected restricted shell command, your changes will be lost.

You can only use the `set_rsc_field` command if the currently selected zone is a classic4 zone is the selected zone. The command does not work in other types of zones.

## Zone type

Classic only

## Syntax

```
set_rsc_field field value
```

## Abbreviation

`srscf`

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
field	string	Required. Specifies the name of the field whose value you want to set.
value		Required. Specifies the value you want to assign to the specified field. The data type depends on the field specified.  In most cases, you can assign an empty string or null value (0) to unset a field value, depending on the data type of the field.

The possible `field` values are:



- **description:** Text describing the restricted shell command.
- **cmd:** The restricted shell command string or strings. You can use wild cards or a regular expression.
- **path:** The path to the command's location. You can use wild cards or a regular expression.
- **form:** An integer that indicates whether the cmd and path strings use wild cards (0) or a regular expression (1).
- **dzsh\_runas:** A list of users and groups that can run this command in a restricted shell environment (dzsh). Users can be listed by user name or UID.
- **keep:** A comma-separated list of environment variables from the current user's environment to keep.
- **del:** A comma-separated list of environment variables from the current user's environment to delete.
- **add:** A comma-separated list of environment variables to add to the final set of environment variables.
- **pri:** An integer that specifies the command priority for the restricted shell command object.
- **umask:** An integer that defines who can execute the command.
- **flags:** An integer that specifies a combination of different properties for the command.
- **createTime:** The time and date this command was created, returned in generalized time format.
- **modifyTime:** The time and date this command was last modified, returned in generalized time format.
- **dn:** The command's distinguished name.

## Setting the cmd and path field values for a restricted command

You can specify the cmd and path strings using wild cards (\*, ?, and !), or as a regular expression. If you specify the cmd and path strings using wild cards, use an asterisk (\*) to match zero or more characters, the question mark (?) to



match exactly one character, or the exclamation mark (!) to negate matching of the specified string.

For both the `cmd` and `path` fields, the `form` field controls whether the specified string is interpreted as a regular expression or as a string that includes wild cards.

## Specifying the environment variables for a restricted command

You can use the `keep`, `del`, and `add` settings to control the environment variables used by the commands specified by the `cmd` string. The `keep` and `del` settings are mutually exclusive. The `keep` field only takes effect if the `flag 16` is included in the setting for the `flag field`. The `del` field only takes effect if the `flag 16` is not included in the setting for the `flag field`.

Any environment variables kept or deleted are in addition to the default set of the user's environment variables that are either retained or deleted. The default set of environment variables to keep is defined in the `dzdo.env_keep` configuration parameter in the `centrifdc.conf` file. The default set of environment variables to delete is defined in the `dzdo.env_delete` configuration parameter in the `centrifdc.conf` file. You can also add environment variables to the final set of environment variables resulting from the `keep` or `del` fields.

## Specifying the restricted command priority

You can use the `pri` field to specify the command priority when there are multiple matches for the restricted shell command object specified by wild cards. If there are multiple commands specified by this restricted shell command object, the restricted shell command with the higher command priority prevails.

## Specifying the umask value for restricted commands

You can use the `umask` field to define who can execute the command. The `umask` field specifies a 3-digit octal value that defines read, write, or execute

• • • • •

permission for owner, group, and other users. The left digit defines the owner execution rights, the middle digit defines the group execution rights, and the right digit defines other execution rights. Each digit is a combination of binary flags, one flag for each right as follows:

- 4 is read
- 2 is write
- 1 is execute

You add these values add together to define the rights available for each entity. For example, a `umask` value of 600 indicates read and write permission (4+2) for the owner, but no permissions for the group or other users. Similarly, a `umask` value of 740 indicates read, write, execute permissions (4+2+1) for the owner, read permissions for the group, but no permissions for other users.

## Specifying restricted command properties using the `flags` field

You can use the **flags** field to define a combination of binary flags, with one flag for each of the following properties:

- 1 to prevent nested command execution. If this flag value is not set, nested command execution is allowed.
- 2 to require authentication with the user's password. You cannot set this flag and the 4 flag simultaneously. If neither 2 nor 4 is set, authentication is not required.
- 4 to require authentication with the run-as user's password  
If you do not set the 2 flag or the 4 flag, authentication is not required.
- 8 to preserve group membership. If this flag value is not set, group membership is not preserved.
- 16 to reset environment variables for the command, deleting the variables specified in the `dzdo.env_delete` parameter and keeping the variables specified in the `keep` field. If this flag is not set, the command removes the unsafe environment variables specified in the `dzdo.env_delete` parameter along with any additional environment variables specified by the `del` field

• • • • •

You add these values together to define the setting for the `flags` field. For example, a `flags` field value of 5 prevents nested command execution and requires authentication using the run-as user's password (1+4).

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_rsc_field description {This is the restricted command
description}
```

This example sets the current restricted shell command `description` field to the "This is the restricted command description" text string.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select the restricted shell command to work with:

- `get_rs_commands` returns a Tcl list of restricted shell commands in the current zone.
- `list_rs_commands` lists to `stdout` the restricted shell commands in the current zone.
- `new_rs_command` creates a new restricted shell command and stores it in memory.
- `select_rs_command` retrieves a restricted shell command from Active Directory and stores it in memory.

After you have a restricted shell command stored in memory, you can use the following commands to work with that restricted shell command:

- `delete_rs_command` deletes the selected command from Active Directory and from memory.
- `get_rsc_field` reads a field value from the currently selected command.



- `save_rs_command` saves the selected command with its current settings to Active Directory.

## set\_rse\_field

Use the `set_rse_field` command to set the value for a specified field in the currently selected restricted shell environment that is stored in memory. The `set_rse_field` command does not set the field value stored in Active Directory for this restricted shell environment.

This command only sets the field value that is stored in memory. You must save the restricted shell environment using the `save_rs_env` command for your changes to take effect in Active Directory. If you select another restricted shell environment or end the ADEdit session before saving the currently selected restricted shell environment, your changes will be lost.

You can only use the `set_rse_field` command if the currently selected zone is a classic4 zone. The command does not work in other type of zones.

## Zone type

Classic only

## Syntax

```
set_rse_field field value
```

## Abbreviation

srsef

## Options

This command takes no options.



## Arguments

This command takes the following argument:

Argument	Type	Description
field	string	Required. Specifies the name of the field whose value you want to set. The only possible value is: <ul style="list-style-type: none"><li>▪ <b>description:</b> Text describing the restricted shell environment.</li></ul>
value	depends on field	Required. Specifies the value to assign to the specified field. In most cases, you can assign an empty string to unset a field value.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_rse_field description {This string is the restricted shell description}
```

This example sets the `description` field for the current restricted shell environment to the “This string is the restricted shell description” text string.

## Related commands

Before you use this command, you must have a currently selected role stored in memory. The following commands enable you to view and select the role to work with restricted shell environments:

- `get_rs_envs` returns a Tcl list of restricted shell environments.
- `list_rs_envs` lists to `stdout` the restricted shell environments.
- `new_rs_env` creates a new restricted shell environment and stores it in memory.



- `select_rs_env` retrieves a restricted shell environment from Active Directory and stores it in memory.

After you have a restricted shell environment stored in memory, you can use the following commands to work with its fields:

- `delete_rs_env` deletes the current restricted shell environment from Active Directory and from memory.
- `get_rse_field` reads a field value from the current restricted shell environment.
- `save_rs_env` saves the restricted shell environment to Active Directory.

## set\_sd\_owner

Use the `set_sd_owner` command to set the owner of a security descriptor (SD). This command requires you to specify the security descriptor in SDDL (security descriptor definition language) form and the security identifier (SID) of the owner. The command sets and returns the updated security descriptor in SDDL form with the new owner.

### Zone type

Not applicable

### Syntax

```
set_sd_owner sddl_string owner_sid
```

### Abbreviation

SSO

### Options

This command takes no options.



## Arguments

This command takes the following arguments:

Argument	Type	Description
sddl_string	string	Required. Specifies a security descriptor in SDDL format.
owner_sid	string	Required. Specifies the security identifier (SID) of the owner to set.

## Return value

This command returns an security descriptor in SDDL format if it runs successfully. The security descriptor contains the new owner set by the command.

## Examples

This example sets a new owner for a security descriptor. The security descriptor is the first long string after the command. The SID of the new owner is the much shorter string at the end of the command (shown in **boldface**).

```
set_sd_owner O:DAG:DAD:AI(A;;;RCWDWOCCDCLCSWRPWPLOCR;;;DA)
(OA;;;CCDC;bf967aba-0de6-11d0-a285-00aa003049e2;;;AO)
(OA;;;CCDC;bf967a9c-0de6-11d0-a285-00aa003049e2;;;AO)
(OA;;;CCDC;bf967aa8-0de6-11d0-a285-00aa003049e2;;;PO)
(A;;;RCLCRPLO;;;AU) (OA;;;CCDC;4828cc14-1437-45bc-9b07-
ad6f015e5f28;;;AO) (OA;CIIOID;RP;4c164200-20c0-11d0-a768-
00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;4c164200-20c0-11d0-a768-
00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
```

.....

```
00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a86-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a9c-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967aba-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RCLCRPLO;;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RCLCRPLO;;bf967a9c-0de6-11d0-
a285-00aa003049e2;RU) (OA;CIIOID;RCLCRPLO;;bf967aba-0de6-
11d0-a285-00aa003049e2;RU) (OA;CIID;RPWPCR;91e647de-d96f-
4b70-9557-d63ff4f3ccd8;;PS)
(A;CIID;SDRCWDWOCCLCSWRPWPDTLOCR;;;EA) (A;CIID;LC;;;RU)
(A;CIID;SDRCWDWOCCLCSWRPWPLOCR;;;BA) S-1-5-21-1076040321-
332654908-468068287-1109
```

This example returns the updated security descriptor:

```
O:S-1-5-21-1076040321-332654908-468068287-1109G:DAD:AI
(A;;RCWDWOCCLCSWRPWPLOCR;;;DA) (OA;;;CCDC;bf967aba-0de6-
11d0-a285-00aa003049e2;;;AO) (OA;;;CCDC;bf967a9c-0de6-11d0-
a285-00aa003049e2;;;AO) (OA;;;CCDC;bf967aa8-0de6-11d0-a285-
00aa003049e2;;;PO) (A;;RCLCRPLO;;;AU) (OA;;;CCDC;4828cc14-1437-
45bc-9b07-ad6f015e5f28;;;AO) (OA;CIIOID;RP;4c164200-20c0-
11d0-a768-00aa006e0529;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RP;4c164200-20c0-11d0-a768-
00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;5f202010-79a5-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;bc0ac240-79a9-11d0-9020-
00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;59ba2f42-79a2-11d0-9020-
00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
```

.....

```
00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)
(OA;CIIOID;RP;037088f8-0ae1-11d2-b422-
00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a86-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967a9c-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RP;b7c69e6d-2cc7-11d2-854e-
00a0c983f608;bf967aba-0de6-11d0-a285-00aa003049e2;ED)
(OA;CIIOID;RCLCRPLO;;4828cc14-1437-45bc-9b07-
ad6f015e5f28;RU) (OA;CIIOID;RCLCRPLO;;bf967a9c-0de6-11d0-
a285-00aa003049e2;RU) (OA;CIIOID;RCLCRPLO;;bf967aba-0de6-
11d0-a285-00aa003049e2;RU) (OA;CIID;RPWPCR;91e647de-d96f-
4b70-9557-d63ff4f3ccd8;;PS)
(A;CIID;SDRCWDWOCCLCSWRPWPDTLOCR;;;EA) (A;CIID;LC;;;RU)
(A;CIID;SDRCWDWOCCLCSWRPWPLOCR;;;BA)
```

## Related commands

The following commands perform actions related to this command:

- `explain_sd` converts an SD in SDDL format to a human-readable form.
- `remove_sd_ace` removes an access control entry (ACE) from an SD.
- `add_sd_ace` adds an access control entry to an SD.

## set\_user\_password

Use the `set_user_password` command to set a new password for an Active Directory user or computer in Active Directory.

## Zone type

Not applicable

## Syntax

```
set_user_password UPN password
```



## Abbreviation

sup

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
UPN	string	Required. Specifies the user principal name (UPN) of the user or computer whose password will be reset.
		Required. Specifies the text string to set as the new password.
password	string	If the string contains characters that might be misinterpreted by ADEdit's Tcl interpreter (\$, for example), enclose the string in braces { } so that all characters are interpreted literally with no substitutions.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_user_password adam.avery@acme.com {B4uC$work}
```

This example sets the password for adam.avery@acme.com to B4uC\$work.

## Related commands

None.

.....

## set\_zone\_computer\_field

Use the `set_zone_computer_field` command to set the value for a specified field in the currently selected zone computer stored in memory. The `set_zone_computer_field` command does *not* set a field value stored in Active Directory for this zone computer.

If you change any fields, you must save the zone computer using the `save_zone_computer` command for your changes to take effect in Active Directory. If you select another zone computer or end the ADEdit session before saving the currently selected zone computer, your changes will be lost.

## Zone type

Classic and hierarchical

## Syntax

```
set_zone_computer_field field value
```

## Abbreviation

szcf

## Options

This command takes no options.

## Arguments

This command takes the following arguments:



---

Argument	Type	Description
field	string	<p>Required. Specifies the name of the field whose value want set. The possible values are:</p> <ul style="list-style-type: none"><li>▪ <b>cpus:</b> Set to a positive integer for the number of CPUs in the computer.</li><li>▪ <b>enabled:</b> Set the value to 1, y, yes, or true if the computer is enabled in the zone or to 0, n, no, or false if the computer is not enabled in the zone. All other values throw an exception.</li><li>▪ <b>licensetype:</b> Specifies the type of license a computer uses. The valid values for this field are <code>server</code> or <code>workstation</code>.</li></ul>
value		<p>Required. Specifies the value to assign to the specified field.</p> <p>In some cases, you can assign a dash (-) to a field to unset the field value. However, this is not supported for all fields or all zone types.</p>

---

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_zone_computer_field cpus 2
```

This example sets the current zone computer's number of CPUs to 2.

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and manage the zone computers:

- `get_zone_computers` returns a Tcl list of the Active Directory names of all zone computers in the current zone.



- `list_zone_computers` lists to stdout the zone computers in the current zone.
- `new_zone_computer` creates a new zone computer and stores it in memory.
- `select_zone_computer` retrieves a zone computer from Active Directory and stores it in memory.

After you have a zone computer stored in memory, you can use the following commands to work with that zone computer:

- `delete_zone_computer` deletes the zone computer from Active Directory and from memory.
- `get_zone_computer_field` reads a field value from the currently selected zone computer.
- `save_zone_computer` saves the zone computer with its current settings to Active Directory.
- `set_zone_computer_field` sets a field value in the currently selected zone computer.

## set\_zone\_field

Use the `set_zone_field` command to set the value for a specified field in the currently selected zone stored in memory. The `set_zone_field` command does *not* set a field value stored in Active Directory for the selected zone.

If you change any fields, you must save the zone using the `save_zone` command for your changes to take effect in Active Directory. If you select another zone or end the ADEdit session before saving the currently selected zone, your changes will be lost.

This command is not applicable if the currently selected zone is a classic-computer zone. You cannot set zone field values for classic-computer zones.

## Zone type

Classic and hierarchical

.....

## Syntax

```
set_zone_field field value
```

## Abbreviation

szf

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
field	string	Required. Specifies the name of the field that you want to set.
		Required. Specifies the value to assign to the specified field.
value		In most cases, you can assign an empty string to unset the field value. For more information about the values set by the zone fields, see the Field value section.

The data type required depends on the `field` you are setting. The possible `field` values are:

- **availableshells:** Sets the list of shells available to choose from when adding new users to the zone.
- **block.parent.zgroup:** Sets the value of the `block.parent.zgroup` field in the zone object's description.
- **cloudurl:** Sets the URL of the cloud instance associated with the selected zone.
- **computers:** Sets the UPN of the computer group assigned to the selected computer role.



- **customAttr:** Sets custom text strings for the zone. This field is only applicable for hierarchical zones.
- **defaultgid:** Sets the default primary group to assign to new users.
- **defaultgecos:** Sets the default GECOS data to assign to new users.
- **defaulthome:** Sets the default home directory to assign to new users.
- **defaultshell:** Sets the default shell to assign to new users.
- **description:** Sets the text string that describes the zone.
- **gidnext:** Sets the next GID to use when auto-assigning GID numbers to new groups.
- **gidreserved:** Sets the GID number or range of numbers (1-100) that are reserved.
- **groupname:** Sets the default group name used for new groups in the zone.
- **nisdomain:** Sets the name of the NIS domain for NIS clients to use.
- **nssvar:** Sets the NSS substitution variable to add to the zone's list of substitution variables.
- **parent:** Sets the distinguished name of the zone's parent zone.
- **sfudomain:** Sets the Windows domain name for the SFU zone.
- **sid2iddomainmap:** Sets the domain ID map for the selected zone. Specify the mapping with a comma-separated key value pairs string. See the examples section for a sample command with this field. Note that the range of domain IDs is 0 to 511. Duplicate mapping entries are not allowed (domain names are not case-sensitive). This field is not supported for auto zones nor classic zones.
- **tenantid:** Returns the Centrify Identity Platform tenant ID for the zone. This field is only applicable for hierarchical zones.
- **uidnext:** Sets the next UID to use when auto-assigning UID numbers to new users.
- **uidreserved:** Sets the UID number or range of numbers (1-100, for example) that are reserved.
- **username:** Sets the default user name used for new users in the zone.

.....

## Return value

This command returns nothing if it runs successfully.

## Examples

The following example sets the computer group associated with the currently selected computer role to `linux_machines` in the domain `acme.com`:

```
set_zone_field computers linux_machines@acme.com
```

The following example sets the parent zone of the current zone to `global` in the domain `acme.com`:

```
szf parent "CN=global,CN=zones,CN=Centrify,CN=Program  
Data,DC=acme,DC=com"
```

The following example sets the domain ID mapping for the selected zone:

```
set_zone_field sid2iddomainmap  
domain0.test=0,domain1.test=1,domain2.test=2
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone to work with:

- [create\\_zone](#) creates a new zone in Active Directory.
- [get\\_zones](#) returns a Tcl list of all zones within a specified domain.
- [select\\_zone](#) retrieves a zone from Active Directory and stores it in memory.

After you have a zone stored in memory, you can use the following commands to work with that zone:

- [delegate\\_zone\\_right](#) delegates a zone use right to a specified user or computer.
- [delete\\_zone](#) deletes the selected zone from Active Directory and memory.



- `get_child_zones` returns a Tcl list of child zones, computer roles, or computer zones.
- `get_zone_field` reads a field value from the currently selected zone.
- `get_zone_nss_vars` returns the NSS substitution variable for the selected zone.
- `save_zone` saves the selected zone with its current settings to Active Directory.

## set\_zone\_group\_field

Use the `set_zone_group_field` command to set the value for a specified field in the currently selected zone group stored in memory. The `set_zone_group_field` command does *not* set a field value stored in Active Directory for the selected zone group.

If you change any fields, you must save the zone group using the `save_zone_group` command for your changes to take effect in Active Directory. If you select another zone group or end the ADEdit session before saving the currently selected zone group, your changes will be lost.

## Zone type

Classic and hierarchical

## Syntax

```
set_zone_group_field field value
```

## Abbreviation

szgf



## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
field	string	<p>Required. Specifies the name of the field that you want to set. The possible values are:</p> <ul style="list-style-type: none"><li>▪ <b>gid</b>: Sets the numeric identifier for the group (GID).</li><li>▪ <b>name</b>: Sets the text string for the group name.</li><li>▪ <b>required</b>: Specifies whether the zone group is required. Set the value to <code>1</code>, <code>y</code>, <code>yes</code>, or <code>true</code> if the group is required in the zone or to <code>0</code>, <code>n</code>, <code>no</code>, or <code>false</code> if the group is not required in the zone. All other values throw an exception.</li></ul> <p>If a group is required, users cannot remove the group from their active set of groups.</p> <p>You can also specify AIX extended attributes as the field to set an extended attribute value for a group. Extended attribute fields start with the <code>aix.</code> prefix. For example, the <code>admin</code> extended attribute can be set by specifying <code>aix.admin</code> as the field.</p>
value		<p>Required. Specifies the value to assign to the specified field. The data type depends on the field specified.</p> <p>In some cases, you can assign a dash (-) to a field to unset the field value. However, this is not supported for all fields or all zone types.</p>

## Return value

This command returns nothing if it runs successfully.



## Examples

The following example sets the current zone group's UNIX group name to `managers`.

```
set_zone_group_field name managers
```

If the current group is on AIX, you can set AIX group extended attributes and values. For example, to identify the current group as an administrative group, you can set the `admin` extended attribute:

```
set_zone_group_field aix.admin true
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select zone groups:

- `get_zone_groups` returns a Tcl list of the Active Directory names of all zone groups in the current zone.
- `list_zone_groups` lists to `stdout` the zone groups in the current zone.
- `new_zone_group` creates a new zone group and stores it in memory.
- `select_zone_group` retrieves a zone group from Active Directory and stores it in memory.

After you have a zone group stored in memory, you can use the following commands to work with that zone group:

- `delete_zone_group` deletes the selected zone group from Active Directory and from memory.
- `get_zone_group_field` reads a field value from the current zone group.
- `save_zone_group` saves the selected zone group with its current settings to Active Directory.

.....

## set\_zone\_user\_field

Use the `set_zone_user_field` command to set the value for a specified field in the currently selected zone user stored in memory. The `set_zone_user_field` command does *not* set a field value stored in Active Directory for this zone user.

If you use ADEdit to change any field, you must save the zone user using the `save_zone_user` command for your changes to take effect in Active Directory. If you select another zone user or end the ADEdit session before saving the currently selected zone user, your changes will be lost.

## Zone type

Classic and hierarchical

## Syntax

```
set_zone_user_field field value
```

## Abbreviation

szuf

## Options

This command takes no options.

## Arguments

This command takes the following arguments:



Argument	Type	Description
		Required. Specifies the name of the field y want set. The possible values are: <ul style="list-style-type: none"><li>▪ <b>uname:</b> Sets the text string to use for the UNIX user name. If you are setting this field in a Service for UNIX (SFU) zone, this name must be unique among all the SFU zones. If you duplicate a user name that exists in another SFU zone, that user will be moved to the currently selected SFU zone when you save the zone user.</li><li>▪ <b>uid:</b> Sets the numeric identifier for the user (UID).</li><li>▪ <b>gid:</b> Sets the numeric identifier for the user’s primary group (GID). Set the value to 0x80000000 to indicate a private group (the user’s UID is used as the GID).</li></ul>
field	string	<ul style="list-style-type: none"><li>▪ <b>gecos:</b> Sets the text string to use for the user’s GECOS field.</li><li>▪ <b>home:</b> Sets the text string that specifies the user’s home directory.</li><li>▪ <b>shell:</b> Sets the text string that specifies the user’s default shell type.</li><li>▪ <b>enabled:</b> Specifies whether user is enabled or not. This field is only valid in classic zones. Set the value to 1, y, yes, or true if the user is enabled in the zone or to 0, n, no, or false if the user is disabled in the zone. All other values throw an exception.</li></ul> <p>You can also specify AIX extended attributes as the field to set an extended attribute value for a zone user.</p>
value		Required. Specifies the value to assign to the specified field. The data type depends on the field specified.  In some cases, you can assign a dash (-) to a field to unset the field value. However, this is not supported for all fields or all zone types.

## Return value

This command returns nothing if it runs successfully.



## Examples

The following example sets the current zone user's UNIX user name to buzz:

```
set_zone_user_field uname buzz
```

This following example sets the current zone user's primary GID to the same value as the user's UID:

```
set_zone_user_field gid 0x80000000
```

If the current zone user is on AIX, you can set extended attributes and values. For example:

```
select_zone_user aixul@acme.com
set_zone_user_field aix.ttys u1,u2,u3
set_zone_user_field aix.fsize 209715
set_zone_user_field aix.core 2097151
set_zone_user_field aix.cpu -1
save_zone_user
```

## Related commands

Before you use this command, you must have a currently selected zone stored in memory. The following commands enable you to view and select a zone user:

- [get\\_zone\\_users](#) returns a Tcl list of the Active Directory names of all zone users in the current zone.
- [list\\_zone\\_users](#) lists to stdout the zone users and their NSS data in the current zone.
- [new\\_zone\\_user](#) creates a new zone user and stores it in memory.
- [select\\_zone\\_user](#) retrieves a zone user from Active Directory and stores it in memory.

After you have a zone user stored in memory, you can use the following commands to work with that zone user:

- [delete\\_zone\\_user](#) deletes the selected zone user from Active Directory and from memory.



- `get_zone_user_field` reads a field value from the currently selected zone user.
- `save_zone_user` saves the selected zone user with its current settings to Active Directory.

## show

Use the `show` command to display the current context of ADEdit. The command shows the domains ADEdit is bound to, the objects that are currently selected, and all available data for each selected object as it is stored in memory.

You should note that the command returns stored object data as it currently exists in memory. If you use ADEdit commands to change objects, but have not yet saved the data back to Active Directory, the information returned by the `show` command will not match the object data stored in Active Directory.

## Zone type

Not applicable

## Syntax

```
show [all|bind|zone|user|computer|assignment|object|group|pamright|dzcommand|nismap|role|license|rse|rscommand|localuser|localgroup]
```

## Abbreviation

None.

## Options

This command takes no options.

.....

## Arguments

This command takes the following argument of type `string`:

```
[all | user | bind | zone | user | computer | assignment |  
object | group | pamright | dzcommand | nismap | role |  
license | rse | rscommand| localuser| localgroup]
```

You can limit the information returned by specifying one of the following arguments. If no argument is supplied, the default is **all**.

- **all** returns the complete context of ADEdit—all of its current bindings and all currently selected objects in memory.
- **bind** returns ADEdit's currently bound domains and the server bound in each domain.
- **zone** returns the currently selected zone.
- **user** returns the currently selected user object.
- **computer** returns the currently selected zone computer.
- **assignment** returns the currently selected role assignment
- **object** returns the currently selected Active Directory object.
- **group** returns the currently selected zone group.
- **pamright** returns the currently selected PAM application right.
- **dzcommand** returns the currently selected UNIX command.
- **nismap** returns the currently selected NIS map.
- **role** returns the currently selected role.
- **license** returns the forest list where valid licenses have been found (it only reports the forests that have been queried).
- **rse** returns the currently selected restricted shell environment.
- **rscommand** returns the currently selected restricted shell command.
- **localuser** returns the currently selected local user.
- **localgroup** returns the currently selected local group.

.....

## Return value

This command returns domain bindings and/or object data, depending on the supplied argument.

## Examples

```
show
```

This example returns information all bound domains and selected objects similar to this:

```
Bindings:
    acme.com: calla.acme.com
Current zone:
    CN=global,CN=Zones,CN=Centrify,CN=Program
Data,DC=acme,DC=com
Current nss user:
    adam.avery@acme.com:adam:10001:10001:%
    {u:samaccountname}:%{home}/%{user}:%{shell}:
```

## Related commands

None.

## sid\_to\_escaped\_string

Use the `sid_to_escaped_string` command to specify a security identifier (SID) and have it converted to an escaped string format that works in an LDAP filter.

## Zone type

Not applicable

.....

## Syntax

```
sid_to_escaped_string sid
```

## Abbreviation

stes

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
sid	string	Required. Specifies a security identifier (SID).

## Return value

This command returns an escaped string form of the supplied security identifier.

## Examples

```
sid_to_escaped_string S-1-5-21-2076040321-3326545908-468068287-1157
```

This example returns an escaped string:

```
\01\05\00\00\00\00\00\05\15\00\00\00\81\dc\bd\7b\xf4\0f\47\c6\b6\27\xe6\1b\85\04\00\00
```



## Related commands

The following commands perform actions related to this command:

- `sid_to_uid` converts an Active Directory security identifier to a user ID (UID).
- `principal_from_sid` searches Active Directory for an security identifier and returns the security principal associated with the security identifier.

## `sid_to_uid`

Use the `sid_to_uid` command to specify a security identifier (SID) of an Active Directory user to look up the Active Directory user in Active Directory. This command converts the user's security identifier to a numeric identifier for the user ID (the UID value). This conversion process is the same process used to generate UIDs for Centrify Express users or when you use Auto Zone to automatically generate UIDs for users.

## Zone type

Not applicable

## Syntax

```
sid_to_uid [-domainidmap] sid
```

## Abbreviation

`stu`

## Options

This command takes the following options:



Option	Description
- domainidmap	<p>Optional. Specifies a domain ID mapping for the selected zone. Before using this field, you must have a selected zone stored in memory. This field is not supported for auto zones nor classic zones.</p> <p>If the selected zone does not already have a domain ID mapping, the UID is generated normally.</p> <p>If the selected zone has a domain ID mapping already and the domain to which this SID belongs exists in the specified domain ID mapping, the UID is generated with the algorithm based on the domain ID mapping.</p> <p>If the selected zone has a domain ID mapping already but the domain to which this SID belongs does not exist in the specified domain ID mapping, the UID is generated normally.</p> <p>For example:</p> <pre>sid_to_uid -domainidmap S-1-5-21-2076040321-3326545908-468068287-1157</pre>

## Arguments

This command takes the following argument:

Argument	Type	Description
sid	string	Required. Specifies a security identifier (SID).

## Return value

This command returns a numeric user ID.

## Examples

This example returns a unique UID for the user: 1874853888

## Related commands

The following commands perform actions related to this command:



- `principal_from_sid` searches Active Directory for an SID and returns the security principal associated with the SID.

## validate\_license

Use the `validate_license` command to specify a path to the Centrify license container and determine if there is a valid license. If there is a valid license, the command stores an indicator in the ADEdit current context. If the command does not find a valid license, it reports an error and exits.

ADEdit requires a valid license before a zone is created. The `create_zone` and `create_computer_role` commands do an implicit search for a valid license. For example, you can call `create_zone` and let it attempt to find the container and validate the license. If that command fails to find a valid license, use `validate_license` to validate the license container from an explicit path.

You can call the `validate_license` command multiple times. Successive indicators take precedence. The command writes separate indicators for each forest—that is, each license is valid for a forest. You can use the `show license` command to see the list of forests that have been found to have a valid license.

Do not call `validate_license` before you bind to the domain.

The `validate_license` context is deleted when ADEdit exits.

## Zone type

Not applicable

## Syntax

```
validate_license path
```

## Abbreviation

v1



## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
path	string	Required. Specifies the path is the license container's distinguished name (DN).

## Return value

This command returns nothing.

## Examples

```
validate_license "CN=Licenses,OU=Centrify,DC=acme,DC=com"
```

This example looks in the `acme.com/Centrify/Licenses` container for a valid license.

## Related commands

The following commands perform actions related to this command:

- `bind` defines the current domain.
- `create_zone` does in implicit validate license during execution.
- `show` with the `Ti cense` option lists all forests that have a valid license.

# ADEdit Tcl procedure library reference

This chapter describes the commands in the `ade_lib` Tcl library. The command descriptions are in alphabetical order. The syntax of each command shows optional elements in [square brackets] and variables in *italics*.

## `add_user_to_group`

Use the `add_user_to_group` command to add an Active Directory user to an Active Directory group.

### Syntax

```
add_user_to_group user group
```

### Options

This command takes no options.

### Arguments

This command takes the following arguments:



Argument	Type	Description
<code>user</code>	string	Required. Specifies the user principal name (UPN) of the Active Directory user to add.
<code>group</code>	string	Required. Specifies the UPN of the Active Directory group to which to add the user.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
add_user_to_group adam.avery@acme.com pubs@acme.com
```

## Related Tcl library commands

The following commands perform actions related to this command:

- `create_aduser` creates a new Active Directory user account and sets its password.
- `create_adgroup` creates a new Active Directory group account and specifies its scope.
- `create_user` creates a new zone user based on an existing Active Directory user, assigns field values to the new user, and saves the new user to Active Directory.
- `create_group` creates a new zone group based on an existing Active Directory group, assigns it a UNIX name and group ID, and saves the new group to Active Directory.
- `remove_user_from_group` removes an Active Directory user from an Active Directory group.



## convert\_msdate

Use the `convert_msdate` command to specify a Microsoft date value from an Active Directory object field such as `pwdLastSet` and convert it into a human-readable form.

### Syntax

```
convert_msdate msdate
```

### Options

This command takes no options.

### Arguments

This command takes the following argument:

Argument	Type	Description
<code>msdate</code>	string	Required. Specifies the Microsoft date value for conversion.

### Return value

This command returns the day of the week, the day of the month, the time of day using a 24-hour clock, the time zone, and the year.

### Examples

```
convert_msdate [get_object_field pwdLastSet]
```

This example returns converted into a format similar to this:

```
Thu Mar 24 14:40:26 PDT 2010
```

.....

The unseen value returned by `get_object_field pwdLastSet` in this example was 12914026824062500, which was converted to a human-readable time and date.

## Related Tcl library commands

None.

## create\_adgroup

Use the `create_adgroup` command to create a new Active Directory group account with a specified distinguished name (DN), `sAMAccountName`, and group scope.

## Syntax

```
create_adgroup dn sam gscope
```

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
<code>dn</code>	string	Required. Specifies the distinguished name of the new group.
<code>sam</code>	string	Required. Specifies the <code>sAMAccountName</code> of the new group.
		Required. Specifies the scope for the new group. The possible values are:
<code>gscope</code>	string	<ul style="list-style-type: none"><li>▪ global</li><li>▪ universal</li><li>▪ local (for Domain local)</li></ul>

.....

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_adgroup {CN=pubs,CN=Users,DC=acme,DC=com} pubs
global
```

This example creates the group `pubs` with a global scope in the Active Directory users container.

```
create_adgroup {CN=ApacheAdmins,OU=Unix
Groups,OU=Centrify,DC=acme,DC=com} pubs global
```

This example creates the group `ApacheAdmins` in the organizational unit `unix groups`, which is in the organizational unit `centrify`.

## Related Tcl library commands

The following commands perform actions related to this command:

- `create_aduser` creates a new Active Directory user account and sets its password.
- `create_user` creates a new zone user based on an existing Active Directory user, assigns field values to the new user, and saves the new user to Active Directory.
- `create_group` creates a new zone group based on an existing Active Directory group, assigns it a UNIX name and group ID, and saves the new group to Active Directory.
- `add_user_to_group` adds an Active Directory user to an Active Directory group.
- `remove_user_from_group` removes an Active Directory user from an Active Directory group.

.....

## create\_aduser

Use the `create_aduser` command to create a new Active Directory user account with a specified distinguished name (DN), user principal name (UPN), sAMAccountName, and password.

### Syntax

```
create_aduser dn upn sam pw ?dname? ?gname? ?spn? ?gecos?
```

### Options

This command takes no options.

### Arguments

This command takes the following arguments:

Argument	Type	Description
dn	string	Required. Specifies the distinguished name of the new user.
upn	string	Required. Specifies the user principal name of the new user.
sam	string	Required. Specifies the sAMAccountName of the new user.
pw	string	Required. Specifies the password for the new user.
dname	string	Optional. Specifies the displayName for the new user.
gname	string	Optional. Specifies the givenName for the new user.
spn	string	Optional. Specifies the servicePrincipalName for the new user.
gecos	string	Optional. Specifies the geocos for the new user.

### Return value

This command returns nothing if it runs successfully.

.....

## Examples

```
create_aduser {CN=ulysses urkham,CN=Users,DC=acme,DC=com}  
ulysses.urkham@acme.com ulysses.urkham {5$6fEr2B}
```

This example creates a new Active Directory user account `ulysses.urkham@acme.com`.

## Related Tcl library commands

- `create_adgroup` creates a new Active Directory group account and specifies its scope.
- `create_user` creates a new zone user based on an existing Active Directory user, assigns field values to the new user, and saves the new user to Active Directory.
- `create_group` creates a new zone group based on an existing Active Directory group, assigns it a UNIX name and group ID, and saves the new group to Active Directory.
- `add_user_to_group` adds an Active Directory user to an Active Directory group.
- `remove_user_from_group` removes an Active Directory user from an Active Directory group.

## create\_assignment

Use the `create_assignment` command to create a new role assignment for a user or group and save it to Active Directory.

## Syntax

```
create_assignment upn role[/zonename] [from] [to]  
[description]
```



## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
upn	string	Required. Specifies the user principal name of the Active Directory user or group to whom to assign the role.
role/ [zonename]	string	Required. Specifies the name of the role to assign and (optional) the name of the zone in which the role is assigned. If the zone name is present, a slash(/) separates the role name and the zone name. If the zone name isn't present, the role assignment occurs in the currently selected zone.
from	string	Optional. Specifies the start date and time for the role assignment. The start date and time can be expressed using the format: yr-mon-day hour:min
to	string	Optional. Specifies the expiration date and time for the role is assignment. The expiration date and time can be expressed using the format: yr-mon-day hour:min
description	string	Optional. Specifies a description of the role assignment.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_assignment ulysse.urkham@acme.com admin/support 0 0  
"Test assignment"
```

This example creates a role assignment for the rights defined in the role "admin" from the "support" zone to the user Ulysses Urkham. The role

.....

assignment is set to start immediately (0) and never expire (0) and has an optional description.

```
create_assignment amy@example.demo mgr {2016-03-31 10:51}
{2016-03-31 12:51}
```

This example creates a role assignment for the rights defined in the role “mgr” from the current zone to the user amy@example.com. This role assignment is set to start at a specific time and expire two hours later and has no description.

## Related Tcl library commands

None.

## create\_dz\_command

Use the `create_dz_command` command to create a new UNIX privileged command in the currently selected zone.

## Syntax

```
create_dz_command dzc cmd ?desc? ?form? ?dzdo_runas? ?dzsh_
runas? ?flags? ?pri? ?umask? ?path? ?selinux_role?
?selinux_type?
```

## Options

This command takes no options.

## Arguments

This command takes the following arguments:



Argument	Type	Description
name	string	Required. Specifies the name to assign to the new UNIX command.
command	string	Required. Specifies the UNIX command string or strings. You can use wild cards or a regular expression.
description	string	Optional. Specifies text describing the UNIX command.
form	integer	Optional. Specifies whether the command and path strings use wild cards (0) or a regular expression (1).
dzdo_ runas	string	Optional. Specifies the list of users and groups that can run this command under dzdo (similar to sudo). Users can be listed by user name or UID.
selinux_ role	string	Optional. Specifies the SELinux role to use when constructing a new security context for command execution.  Note that selinux_role is only supported on Red Hat Enterprise Linux systems and effective only on systems with SELinux enabled and joined to a hierarchical zone.
selinux_ type	string	Optional. Specifies the SELinux type to use when constructing a new security context for command execution.  Note that selinux_type is only supported on Red Hat Enterprise Linux systems and effective only on systems with SELinux enabled and joined to a hierarchical zone.
dzsh_ runas	string	Optional. Specifies the list of users and groups that can run this command in the restricted shell environment (dzsh). Users can be listed by user name or UID.
flags	integer	Optional. Specifies an integer that defines a combination of different properties for the command.  For more information about setting this field, see <a href="#">set_dzc_field</a> .
pri	integer	Optional. Specifies the command priority for the restricted shell command object.  For more information about setting this field, see <a href="#">set_dzc_field</a> .



Argument	Type	Description
umask	integer	Optional. Specifies an integer that defines who can execute the command. For more information about setting this field, see <a href="#">set_dzc_field</a> .
path	string	Optional. Specifies the path to the command's location. You can use wild cards, a regular expression, or one of the following keywords: <ul style="list-style-type: none"><li>■ USERPATH to set to the command path to the equivalent of the Standard user path option.</li><li>■ SYSTEMPATH to set to the path to the equivalent of the Standard system path option.</li><li>■ SYSTEMSEARCHPATH to set to the path to the equivalent of the System search path option.</li></ul> If you don't specify this argument, the default is USERPATH.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_dz_command testvi vi {Test UNIX command vi} {}  
{sfapps:perez,cody} {} {16}
```

## Related Tcl library commands

None.

## create\_group

Use the `create_group` command to create a new zone group for the currently selected zone. This command creates the new group based on an existing Active Directory group. It also assigns the new group a new UNIX profile that includes the UNIX group name and the UNIX group numeric identifier (GID).



## Syntax

```
create_group upn name gid ?req?
```

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
upn	string	Required. Specifies the user principal name of the Active Directory group to use as the basis for the new zone group.
name	string	Required. Specifies the UNIX group name of the new zone group. For hierarchical zones only, specifying “-” unsets the name value.
gid	string	Required. Specifies the UNIX group ID to assign to the new zone group. For hierarchical zones only, specifying “-” unsets the gid value.
req	string	Optional. Specifies whether the zone group is required. Set the value to 1, y, yes, or true if the group is required in the zone or to 0, n, no, or false if the group is not required in the zone. All other values throw an exception. If a group is required, users cannot remove the group from their active set of groups.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_group pubs@acme.com pubs 1094
```



## Related Tcl library commands

The following commands perform actions related to this command:

- `create_aduser` creates a new Active Directory user account and sets its password.
- `create_adgroup` creates a new Active Directory group account and specifies its scope.
- `create_user` creates a new zone user based on an existing Active Directory user, assigns field values to the new user, and saves the new user to Active Directory.
- `add_user_to_group` adds an Active Directory user to an Active Directory group.
- `remove_user_from_group` removes an Active Directory user from an Active Directory group.

## create\_nismap

Use the `create_nismap` command to create a new NIS map in the currently selected zone.

## Syntax

```
create_nismap map key:value comment
```

## Options

This command takes no options.

## Arguments

This command takes the following arguments:



Argument	Type	Description
map	string	Required. Specifies the name of the new NIS map
key	string	Required. Specifies the key of the NIS map entry.
value	string	Required. Specifies the value of the NIS map entry.
comment	string	Required. Specifies the comment for the NIS map entry.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_nismap animals {{cat:1 {The cat says "Mew\!".}}  
{cow:1 {The cow says "Moo\!".}}}
```

## Related Tcl library commands

None.

## create\_pam\_app

Use the `create_pam_app` command to create a new PAM application access right in the currently selected zone.

## Syntax

```
create_pam_app name application description
```

## Options

This command takes no options.



## Arguments

This command takes the following arguments:

Argument	Type	Description
name	string	Required. Specifies the name to assign to the new PAM application access right.
application	string	Required. Specifies the name of the PAM application that is allowed to use the <code>adclient</code> PAM authentication service. The name can be literal, or it can contain <code>?</code> or <code>*</code> wild card characters to specify multiple applications.  Note that in a classic zone, setting the <b>application</b> field changes the name of the PAM application right. For example, assume you create a new PAM application right in a classic zone using a command like this: <pre>create_pam_app myftp newftp "Sample PAM FTP application"</pre> The PAM application right itself will be renamed as <code>newftp</code> : <pre>list_pam_apps newftp : Sample PAM FTP application</pre> Therefore, in a classic zone, you should always specify the same string for the name and application arguments. In a hierarchical zone, you can specify different strings for the arguments.
description	string	Optional. Specifies the text describing the PAM application.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_pam_app testvi vi {Test UNIX command vi}
```

## Related Tcl library commands

None.



## create\_role

Use the `create_role` command to create a new role definition in the currently selected zone.

### Syntax

```
create_role name description sysrights pamrights cmdrights  
allowlocal rsend visible
```

### Options

This command takes no options.

### Arguments

This command takes the following arguments:

Argument	Type	Description
name	string	Required. Specifies the name to assign to the new role.
description	string	Specifies the text string that describes the role.
sysrights	integer	Specifies the system rights granted to the role. This value is an integer that represents a combination of binary flags, one for each system right. This field is not applicable in classic zones.
pamrights [/zonename]	string	Specifies the PAM application rights to add to the currently selected role.  If the PAM application right that you want to add is defined in the current zone, the <i>zonename</i> argument is optional. If the PAM application right is defined in a zone other than the currently selected zone, the <i>zonename</i> argument is required to identify the specific PAM application right to add.
cmdrights [/zonename]	string	Specifies the UNIX command rights to add to the currently selected role.  If the UNIX command right that you want to add is defined in the current zone, the <i>zonename</i> argument is optional. If the UNIX command right is defined in a zone other than the currently selected



Argument	Type	Description
		zone, the <i>zonename</i> argument is required to identify the specific UNIX command right to add.
allowlocal	Boolean	Specifies whether local users can be assigned to the role. If this argument is specified, local users can be assigned to the role. This argument is only applicable if the zone is a hierarchical zone.
rsenv	string	Specifies a restricted shell environment for the role you are creating. This argument is only applicable if the zone is a classic zone.
visible	Boolean	Specifies whether the account profiles for Active Directory users in the role are visible on computers in the zone. This argument is only applicable if the zone is a hierarchical zone.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_role dba {Database admins - US} 11 {{oracle} {ftp}}  
{{testvi} {ora-stp}}
```

## Related Tcl library commands

None.

## create\_rs\_command

Use the `create_rs_command` command to create a new restricted shell command for the currently selected restricted shell environment.



## Syntax

```
create_rs_command rsc_name cmd description form dzsh_runas  
flags pri umask path
```

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
rsc_name	string	Required. Specifies the name of the restricted shell command.
cmd	string	Required. Specifies the restricted shell command string or strings. You can use wild cards or a regular expression.
description	string	Optional. Specifies the text describing the restricted shell command.
form	integer	Optional. Indicates whether the cmd and path strings use wild cards (0) or a regular expression (1).
dzsh_runas	string	Optional. Specifies the list of users and groups that can run this command in a restricted shell environment (dzsh). Users can be listed by user name or UID.
flags	string	Optional. Specifies an integer that specifies a combination of different properties for the command. For more information about setting this field, see <a href="#">set_rsc_field</a> .
pri	integer	Optional. Specifies the command priority for the restricted shell command object. For more information about setting this field, see <a href="#">set_rsc_field</a> .



Argument	Type	Description
umask	integer	Optional. Specifies an integer that defines who can execute the command. For more information about setting this field, see <a href="#">set_rsc_field</a> .
path	string	Optional. Specifies the path to the restricted command. You can use wild cards, a regular expression, or one of the following keywords: <ul style="list-style-type: none"><li>■ <code>USERPATH</code> to set to the command path to the equivalent of the Standard user path option.</li><li>■ <code>SYSTEMPATH</code> to set to the path to the equivalent of the Standard system path option.</li><li>■ <code>SYSTEMSEARCHPATH</code> to set to the path to the equivalent of the System search path option.</li></ul> If you don't specify this argument, the default is <code>USERPATH</code> .

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_rs_command test_id id {Sample restricted command description}
```

## Related Tcl library commands

The following commands perform actions related to this command:

- [create\\_rs\\_env](#) creates a new restricted shell environment.

## create\_rs\_env

Use the `create_rs_env` command to create a new restricted shell environment for the currently selected zone.

.....

## Syntax

```
create_rs_env rse_name rse_description
```

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
rse_name	string	Required. Specifies the name of the new restricted shell environment.
rse_description	string	Optional. Specifies the description for the new restricted shell environment.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_rs_env restrictedenv "This is a restricted shell environment"
```

## Related Tcl library commands

The following commands perform actions related to this command:

- `create_rs_command` creates a new restricted shell command.

.....

## create\_user

Use the `create_user` command to create a new zone user for the currently selected zone. This command creates the new user based on an existing Active Directory user. It also assigns the new user a new UNIX profile that includes the user name, user ID, primary group ID, GECOS data, home directory, shell type, and role (or in classic zones whether the user is enabled or disabled).

You can assign the new user a role in a non-classic zone or you can enable or disable the new user in a classic zone. In a non-classic zone, `create_user` uses whatever role you specify to create a new role assignment object that links the new zone user to the specified role.

## Syntax

```
create_user UPN uname uid gid gecos home shell role
```

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
UPN	string	Required. Specifies the user principal name of the Active Directory user to use as the basis for the new zone user.
uname	string	Required. Specifies the user name of the new zone user. For hierarchical zones, you can specify a dash (-) for this argument if you don't want to set the user name.
uid	string	Required. Specifies the user ID for the new zone user. For hierarchical zones, you can specify a dash (-) for this argument if you don't want to set the user ID.



Argument	Type	Description
gid	string	Required. Specifies the group ID for the new zone user. For hierarchical zones, you can specify a dash (-) for this argument if you don't want to set the group ID.
gecos	string	Required. Specifies the GECOS value (new user account information) for the new zone user. For hierarchical zones, you can specify a dash (-) for this argument if you don't want to set the GECOS value. You can't set the GECOS value if the currently selected zone is a classic zone.
home	string	Required. Specifies the home directory for the new zone user. For hierarchical zones, you can specify a dash (-) for this argument if you don't want to set the home directory.
shell	string	Required. Specifies the shell type for the new zone user. For hierarchical zones, you can specify a dash (-) for this argument if you don't want to set the shell type.
role	string or Boolean value	Required. For classic zones, this argument determines whether to enable or disable the new zone user. A value of 1, Y, or y enables the user. Any other value disables the user. For hierarchical zones, this argument identifies the role to assign to the new zone user. You can specify a dash (-) for this argument if you don't want to set the role. However, a role must be assigned before the new zone user has access to computers in hierarchical zones.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
create_user ulysses.urkham@acme.com ulysses 1005 - - %  
{home}/%{user} %{shell} -
```

This example creates a zone user “ulysses” based on the Active Directory user `ulysses.urkham@acme.com`. It sets a UID, does not set a GID or GECOS value

• • • • •

by using dashes, sets home and shell values, and does not set a role value (specified by using a dash).

## Related Tcl library commands

- [create\\_aduser](#) creates a new Active Directory user account and sets its password.
- [create\\_adgroup](#) creates a new Active Directory group account and specifies its scope.
- [create\\_group](#) creates a new zone group based on an existing Active Directory group, assigns it a UNIX name and group ID, and saves the new group to Active Directory.
- [add\\_user\\_to\\_group](#) adds an Active Directory user to an Active Directory group.
- [remove\\_user\\_from\\_group](#) removes an Active Directory user from an Active Directory group.

## decode\_timebox

Use the `decode_timebox` command to convert an internal timebox value that defines when a role is enabled or disabled into a format that can be evaluated. The command converts the internal hexadecimal value for a role timebox to a hexadecimal timebox value format as described in [Timebox value format](#)

The command returns a 168-bit value in hexadecimal format that delineates the hours of the week from midnight Sunday to 11 PM Saturday in order from most-significant bit to least-significant bit. If a bit is set to 1, its corresponding hour is enabled for the role. If set to 0, its corresponding hour is disabled.

This command is useful for deciphering the value returned by the [get\\_role\\_field](#) for the timebox field.

## Syntax

```
decode_timebox strTimeBox
```

.....

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
strTimeBox	hex	A 42-digit hexadecimal timebox value. A value of zero disables all hours of the week. A value of FF enables all hours of the week.

## Return value

This command returns a decoded hexadecimal value that is the timebox value for a role.

## Examples

```
>select_role test1
>get_role_field timebox
FFF7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
>package require ade_lib
1.0
>decode_timebox [grf timebox]
```

This example returns the decoded 42 hexadecimal that indicates the role is disabled from midnight to one on Sunday:

```
7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

## Related Tcl library commands

The following commands perform actions related to this command:



- `encode_timebox` converts a readable timebox value to an internal timebox format.
- `modify_timebox` defines an hour of the week and enables or disables that hour in the timebox value.

## encode\_timebox

Use the `encode_timebox` command to convert a human-readable timebox value that defines the when a role is enabled or disabled to an internal timebox value format.

The command converts the hexadecimal timebox value format described in [Timebox value format](#) to the internal hexadecimal value for a role. The command accepts a 168-bit value in hexadecimal format that delineates the hours of the week from midnight Sunday to 11 PM Saturday from most-significant bit to least-significant bit. If a bit is set to 1, its corresponding hour is enabled for the role. If set to 0, its corresponding hour is disabled.

This command is useful for setting the timebox field with the [set\\_role\\_field](#) command.

## Syntax

```
encode_timebox strTimeBox
```

## Options

This command takes no options.

## Arguments

This command takes the following arguments:



Argument	Type	Description
strTimeBox	hex	A 42-digit hexadecimal timebox value. A value of zero disables all hours of the week. A value of FF enables all hours of the week.

## Return value

This command returns a decoded hexadecimal value that is the timebox value for a role.

## Examples

```
>package require ade_lib
>set tb 7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
>encode_timebox $tb
```

This example returns the encoded 42 hexadecimal that indicates the role is disabled from midnight to one on Sunday:

```
FFF7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

## Related ade\_lib Tcl library commands

The following commands perform actions related to this command:

- [decode\\_timebox](#) converts an internal timebox value to a decipherable format.
- [modify\\_timebox](#) defines an hour of the week and enables or disables that hour in the timebox value.

## explain\_groupType

Use the `explain_groupType` command to convert a `groupType` value from an Active Directory object field into human-readable form.

.....

## Syntax

```
explain_groupType gt
```

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
gt	string	Required. A groupType value for conversion.

## Return value

This command returns a hexadecimal version of the supplied value followed by the names of any flags that are set in the value.

## Examples

```
explain_groupType [get_object_field groupType]
```

This example returns:

```
80000004 DOMAIN_LOCALSECURITY
```

The unseen value returned by `get_object_field groupType` in this example was `-2147483644`, which was converted to the hexadecimal value `80000004` and the name of the set flag `DOMAIN_LOCALSECURITY`.

## Related Tcl library commands

The following commands perform actions related to this command:



- `explain_trustAttributes` converts a `trustAttributes` value from an Active Directory object into human-readable form.
- `explain_trustDirection` converts a `trustDirection` value from an Active Directory object into human-readable form.
- `explain_userAccountControl` converts a `userAccountControl` value from an Active Directory object into human-readable form.

## explain\_ptype

Use the `explain_ptype` command to translate the account type for a role assignment into a descriptive text string.

### Syntax

```
explain_ptype pt
```

### Options

This command takes no options.

### Arguments

This command takes the following argument:

Argument	Type	Description
<code>pt</code>	string	Required. Specifies the <code>ptype</code> value returned for a role assignment that you want to convert to a text string.

### Return value

This command returns a text string that describes the type of account associated with a role assignment.

.....

## Examples

```
select_role_assignment "lulu@acme.test/UNIX Login"  
get_role_assignment_field ptype  
a  
explain_ptype a
```

This example returns:

```
All AD users
```

The following table summarizes the descriptive names for different account types that can be associated with a role assignment:

<b>Account type</b>	
Local UNIX user	#
Local UNIX group	%
Local Windows User	\$
Local Windows Group	:
All AD users	a
All Unix users	x
All Windows users	w

## explain\_trustAttributes

Use the `explain_trustAttributes` command to convert a `trustAttributes` value from an Active Directory object field into human-readable form.

## Syntax

```
explain_trustAttributes ta
```

## Options

This command takes no options.

.....

## Arguments

This command takes the following argument:

Argument	Type	Description
ta	string	Required. A trustAttributes value for conversion.

## Return value

This command returns a hexadecimal version of the supplied value followed by the names of any flags that are set in the value.

## Examples

```
explain_trustAttributes [get_object_field trustAttributes]
```

This example returns:

```
8 FOREST_TRANSITIVE
```

The unseen value returned by `get_object_field trustAttributes` in this example was 8, which was converted to the hexadecimal value 8 and the name of the set flag `DOMAIN_LOCALSECURITY`.

## Related Tcl library commands

The following commands perform actions related to this command:

- `explain_groupType` converts a `groupType` value from an Active Directory object into human-readable form.
- `explain_trustDirection` converts a `trustDirection` value from an Active Directory object into human-readable form.
- `explain_userAccountControl` converts a `userAccountControl` value from an Active Directory object into human-readable form.

.....

## explain\_trustDirection

Use the `explain_trustDirection` command to convert a `trustDirection` value from an Active Directory object field into human-readable form.

### Syntax

```
explain_trustDirection td
```

### Options

This command takes no options.

### Arguments

This command takes the following argument:

Argument	Type	Description
<code>td</code>	string	Required. A <code>trustDirection</code> value for conversion.

### Return value

This command returns the English version of the trust direction specified by the `trustDirection` value.

### Examples

```
explain_trustDirection [get_object_field trustDirection]
```

This example returns:

```
two-way
```



## Related Tcl library commands

The following commands perform actions related to this command:

- `explain_groupType` converts a `groupType` value from an Active Directory object into human-readable form.
- `explain_trustAttributes` converts a `trustAttributes` value from an Active Directory object into human-readable form.
- `explain_userAccountControl` converts a `userAccountControl` value from an Active Directory object into human-readable form.

## explain\_userAccountControl

Use the `explain_userAccountControl` command to convert a `userAccountControl` value from an Active Directory object field into a human-readable form.

## Syntax

```
explain_userAccountControl uac
```

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>uac</code>	string	Required. A <code>userAccountControl</code> value for conversion.

• • • • •

## Return value

This command returns a hexadecimal version of the supplied value followed by the names of any flags that are set in the value.

## Examples

```
explain_userAccountControl [get_object_field  
userAccountControl]
```

returns:

```
10200 ADS_UF_NORMAL_ACCOUNT ADS_UF_DONT_EXPIRE_PASSWD
```

The unseen value returned by `get_object_field userAccountControl` in this example was 66048, which was converted to the hexadecimal value 10200 and the name of the set flags `ADS_UF_NORMAL_ACCOUNT` and `ADS_UF_DONT_EXPIRE_PASSWD`.

## Related Tcl library commands

The following commands perform actions related to this command:

- `explain_groupType` converts a `groupType` value from an Active Directory object into human-readable form.
- `explain_trustAttributes` converts a `trustAttributes` value from an Active Directory object into human-readable form.
- `explain_trustDirection` converts a `trustDirection` value from an Active Directory object into human-readable form.

## get\_all\_zone\_users

Use the `get_all_zone_users` command to check Active Directory and return a list of zone users defined within the specified zone and all of its parent zones. If executed in a script, this command does not output its list to stdout, and no output appears in the shell where the script is executed.



Note that this command does *not* use the currently selected zone to find its list of users. It uses instead the zone specified as an argument for the command. It ignores the currently selected zone. The selected zone remains the selected zone after the command executes.

## Syntax

```
get_all_zone_users [-upn] zone_DN
```

## Abbreviation

None.

## Options

This command takes the following option:

Option	Type	Description
-upn	string	Return user names in the Tcl list as universal principal names (UPNs).

## Arguments

This command takes the following argument:

Argument	Type	Description
zone_DN	string	Required. The distinguished name (DN) of the zone for which to return users.

## Return value

This command returns a Tcl list of zone users defined in the currently selected zone and all of its parent zones. Each entry in the list is in the format sAMAccountName@domain. If a zone user is an orphan user (its corresponding Active Directory user no longer exists), the user is listed by its security identifier (SID) instead of the sAMAccountName.

• • • • •

If the `-upn` option is present, each entry in the returned Tcl list is a universal principal name (UPN).

## Examples

```
get_all_zone_users engineering
```

The example returns the list of zone users:

```
adam.avery@acme.com brenda.butler@acme.com  
chris.carter@acme.com dave.douglas@acme.com  
elliott.evans@acme.com
```

## Related Tcl library commands

The following commands perform actions related to this command:

- `create_user` creates a new zone user and user profile based on a specified Active Directory user.
- `create_group` creates a new zone group and group profile based on a specified Active Directory group.
- `get_effective_groups` returns a Tcl list of groups to which a specified user belongs.

## get\_effective\_groups

Use the `get_effective_groups` command to return the list of effective groups from current zone up the zone hierarchy. Only groups who have a complete profile—whether defined in the current zone or inherited from a parent zone—are included.

The command supports hierarchical zone and classic zones. For classic zones, the command starts from current zone. For hierarchical zones, you can start the search for effective groups at the computer level by specifying the `-hostname` option.

You can use the `adinfo` command to return the computer name.

.....

## Syntax

```
get_effective_groups [-hostname computer_name]
```

## Options

This command takes the following option:

Option	Type	Description
-hostname	string	Specifies the name of the computer to start the search at the computer or computer role level if you run the command in a hierarchical zone with computer-level overrides or computer roles. If you don't specify this option, the search starts in the current zone and computer roles are ignored.

## Return value

This command returns a Tcl list of groups with complete profiles in the currently selected zone and all of its parent zones.

## Example

```
get_effective_groups -hostname centos7.ajax.com
```

The example returns the list of effective groups starting at the computer level for the computer named `centos7.ajax.com`.

## get\_effective\_users

Use the `get_effective_users` command to return the list of effective users from current zone up the zone hierarchy. Only users who have a complete profile—whether defined in the current zone or inherited from a parent zone—are included. Similarly, only users who have a role assignment in the current zone or inherited from a parent zone are included.

• • • • •

The command supports hierarchical zone and classic zones. For classic zones, the command starts from current zone. For hierarchical zones, you can start the search for effective users at the computer level by specifying the `-hostname` option.

## Syntax

```
get_effective_users [-hostname computer_name]
```

## Options

This command takes the following option:

Option	Type	Description
<code>-hostname</code>	string	Specifies the name of the computer to start the search at the computer or computer role level if you run the command in a hierarchical zone with computer-level overrides or computer roles. If you don't specify this option, the search starts in the current zone and computer roles are ignored.

## Return value

This command returns a Tcl list of users with complete profiles and at least one role assignment in the currently selected zone and all of its parent zones.

## Example

```
get_effective_users -hostname centos7.ajax.com
```

The example returns the list of effective users starting at the computer level for the computer named `centos7.ajax.com`.

.....

## get\_user\_groups

Use the `get_user_groups` command to check Active Directory for a specified user and return a list of the groups to which the user belongs. If executed in a script, this command does not output its list to stdout, and no output appears in the shell where the script is executed.

## Syntax

```
get_user_groups [-dn] [-z] user_DN|user_UPN
```

## Abbreviation

None.

## Options

This command takes the following options:

Option	Description
-dn	Return groups in the Tcl list as distinguished names (DNs) instead of user principal names (UPNs).
-z	Restricts the Tcl list of groups to groups that belong to the current zone.

## Arguments

This command takes the following argument:

Argument	Type	Description
user_DN user_UPN	string	Required. The user whose groups to return. This argument may specify the user with a distinguished name (DN) or a user principal name (UPN).



## Return value

This command used without options returns a Tcl list of all groups listed in Active Directory to which the specified user belongs. Each entry in the list is the user principal name (UPN) of a group that you can use to look up that group.

If the `-dn` option is set, the Tcl list uses distinguished names (DNs) for groups.

If the `-z` option is set, the Tcl list is restricted to groups that belong to the currently selected zone.

Note that the command will not return groups for domains that aren't currently bound to ADEdit. If the command finds one or more groups outside of the currently bound domains, it will return a "no binding" message for each unbound domain in which it finds a user's group.

## Examples

```
get_user_groups fred.forth@acme.com
```

This example returns a list of groups:

```
poweradmins@acme.com auditors@acme.com
```

## Related Tcl library commands

The following commands perform actions related to this command:

- [create\\_group](#) creates a new zone group and group profile based on a specified Active Directory group.
- [create\\_user](#) creates a new zone user and user profile based on a specified Active Directory user.
- [get\\_all\\_zone\\_users](#) returns a Tcl list of zone users for the specified zone and all of its parent zones.



## get\_user\_role\_assignments

Use the `get_user_role_assignments` command to retrieve all of the role assignments in the current zone for a specified user. This command returns all of the role assignments from the groups to which the user belongs and the role assignments assigned directly to the user account.

The command checks Active Directory for the user you specify, identifies the groups that the user is a member of, then returns all the role assignments for the list of groups the user is a member and that have been specifically assigned to the user account, including any user role assignments made in computer roles for the currently selected zone.

### Syntax

```
get_user_role_assignments [-visible] [-hostname hostname]  
user_DN
```

### Abbreviation

None.

### Options

This command takes the following option:

Option	Description
-visible	Specifies that you want to return only visible role assignments in the zone. Use this option to return role assignments for the roles that are identified as visible. This option is only applicable in hierarchical zones.
-hostname	Specifies the computer name to search for role assignments to the user in computer roles. If you set this option, the command checks for computer role assignments in the currently selected zone.

.....

## Arguments

This command takes the following argument:

Argument	Type	Description
user_DN	string	Required. Specifies the user whose role assignments you want to return. You can use this argument to specify the distinguished name (DN) for a user or a group.

## Return value

This command returns a list of all role assignments for the specified Active Directory user in the currently selected zone.

Note that the command does not return role assignments for all zones where the user might be assigned a role.

## Examples

```
select_zone
"cn=northamerica,cn=zones,ou=centrify,dc=pistolas,dc=org"

get_user_role_assignments
"cn=amy.adams,cn=users,dc=pistolas,dc=org"
```

This example returns a list of groups:

```
{amy.adams@pistolas.org/UNIX Login/northamerica}
{adm-sf@pistolas.org/Root/sanfrancisco}
{apps@pistolas.org/demos/seattle}
```

## Related Tcl library commands

The following commands perform actions related to this command:

- [get\\_all\\_zone\\_users](#) returns a Tcl list of zone users for the specified zone and all of its parent zones.
- [get\\_effective\\_groups](#) returns a list of the groups to which the user belongs.

.....

## list\_zones

Use the `list_zones` command to list the zones within a specified domain along with information about each zone. If executed in a script, this command outputs its list to `stdout` so that the output appears in the shell where the script is executed. The command does not return a Tcl list back to the executing script. Use the ADEdit command `get_zones` to return a Tcl list.

## Syntax

```
list_zones domain
```

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>domain</code>	string	Required. The name of the domain in which to list zones.

## Return value

This command returns a list to `stdout` of the zones within the specified domain. Each entry in the list contains:

- The zone's distinguished name (DN)
- The zone type: `tree` (supported in Centrify Server Suite 2012 or later), `classic3` or `classic4`
- The schema used in the zone

Each entry component is separated from the next by a colon (:).

.....

## Examples

```
list_zones
```

This example returns a list of zones similar to this:

```
{CN=default,CN=Zones,CN=Centrify,DC=acme,DC=com} : classic4
: std
{CN=cz1,CN=Zones,CN=Centrify,DC=acme,DC=com} : tree : std
{CN=cz2,CN=Zones,CN=Centrify,DC=acme,DC=com} : tree : std
{CN=global,CN=Zones,CN=Centrify,DC=acme,DC=com} : tree :
rfc
```

## Related Tcl library commands

The following commands perform actions related to this command:

- [create\\_assignment](#) creates a new role assignment and saves it to Active Directory.
- [precreate\\_computer](#) creates a zone profile and, if necessary, a new Active Directory computer account.

## lmerge

Use the `lmerge` command to merge and sort the specified lists. You specify the lists to merge as arguments. You must enclose the list commands you want to merge in square brackets.

## Syntax

```
lmerge [list1] [list2] [list[...]]
```

## Options

This command takes no options.

.....

## Arguments

This command takes the following arguments:

Argument	Type	Description
[list1]	string	Specifies the list command that return the information you want to include first in the merged results.
[list2]	string	Specifies the list command that return the information you want to include second in the merged results.
[list[...]]	string	Specifies any additional list commands that return information you want to include in the merged results.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
lmerge [list_zone_users] [list_zone_computers] [list_roles]
```

This example returns a merged list of zone users, zone computers, and zone roles similar to this:

```
fred@pistolas.org:fred:580398:648:%{u:displayName}:%  
{home}/%{user}:%{shell}:  
lane@pistolas.org:lane:580397:648:%{u:displayName}:%  
{home}/%{user}:%{shell}:  
maya@pistolas.org:maya:580320:648:%{u:displayName}:%  
{home}/%{user}:%{shell}:  
ubul$@pistolas.org: cpus(1) agentVersion(CentrifyDC 5.2.0):  
ubul.pistolas.org  
nic3$@pistolas.org: cpus(2) agentVersion(CentrifyDC 5.2.0):  
nic3.pistolas.org  
Rescue - always permit login  
listed  
UNIX Login  
UnixAdminRights  
Windows Login
```

.....

You can specify the list arguments using full command names or abbreviations. For example:

```
lmerge [lszc] [lspa]
ubu1$@pistolas.org: cpus(1) agentVersion(CentrifyDC 5.2.0):
ubu1.pistolas.org
nic3$@pistolas.org: cpus(2) agentVersion(CentrifyDC 5.2.0):
nic3.pistolas.org
dzssh-all/Headquarters : dzssh-* : All of ssh services
login-all/Headquarters : * : Predefined global PAM
permission. Do not delete.
```

## Related Tcl library commands

None.

## modify\_timebox

Use the `modify_timebox` command to modify a timebox value that defines the hours of a week when a role is enabled or disabled. The command defines an hour of the week and then enables or disables that hour in the timebox value. This command is very useful in the `set_role_field` AEdit command when setting the timebox field.

Execute this command multiple times on a timebox value to set more than one hour in the value.

For more information about the timebox value format, read the [Timebox value format](#).

## Syntax

```
modify_timebox strTimeBox day hour avail
```

## Options

This command takes no options.





- `encode_timebox` converts a readable timebox value to an internal timebox format.

## precreate\_computer

Use the `precreate_computer` command to create a zone profile for a computer in Active Directory before using the `adjoin` command to join the domain. The zone profile—a `serviceConnectionPoint` (`scp`) object—is usually created by the `adjoin` command when a computer joins the domain. In some cases, however, creating the zone profile before joining is useful. For example, preparing the computer object before joining enables you to check that you have user profiles and role assignments correctly defined before you join UNIX computers to zones. Verifying this information before the join operation helps to ensure a smooth migration without disrupting users' access to files or applications.

The zone profile is part of an Active Directory computer object. If an Active Directory computer object doesn't exist, `precreate_computer` can create one and then add the zone profile to the new Active Directory computer object. The zone profile is created in ADEdit's currently selected zone. You can also use the `precreate_computer` command to specify a container where Active Directory will store the new Active Directory computer object.

You can use the `precreate_computer` command to create a service connection point for a new or existing Active Directory computer object. You can also use the command to create a computer-specific zone for machine-level zone overrides (in essence a one-computer zone) for the precreated computer. You should note that performing these tasks requires access to the global catalog by default. You can intentionally skip the global catalog search if you know the service connection point you are creating is unique in the forest. However, skipping the global catalog search might prevent you from joining the computer to the domain if there is a conflict.

The `precreate_computer` command also sets the Active Directory computer object's password and permissions when creating a zone profile. The password is the computer's host name in lower case. The permissions the computer object has are:

- Read and Write permissions to the `operatingSystemServicePack`, `operatingSystem`, and `operatingVersion` attributes of the computer

.....

object.

- Read permission for the `userAccountControl` attribute of the computer object.
- Validate write to the `servicePrincipalName` and `dnsHostName` attributes.

You can use `precreate_computer` to specify a DNS name for the precreated computer and one or more trustees for the precreated computer. Each trustee can be either a user or a group, and has the rights needed to join the computer to the precreated computer account using `adjoin`.

Use the `precreate_computer` command option, `enctype`, to specify encryption types.

The `precreate_computer` command is similar to using `adjoin -precreate`, but provides more options and flexibility. You can also precreate computer accounts using Access Manager. For more information about precreating computer accounts, See the *Administrator's Guide for Linux and UNIX*.

## Syntax

```
precreate_computer samaccount@domain[-ad] [-scp] [-czone]
[-all] [-container rdn]
[-dnsname dnsname] [licensetype type] [-trustee upn[-
trustee upn] ...] [-nogc]
[-stype spn [-stype spn] ...] [-enctype type [-enctype
type] ...]
```

## Options

This command takes the following options:

Option	Description
-ad	Creates an Active Directory computer object. <code>precreate_computer</code> won't create an Active Directory computer object if it already exists for the computer specified by the argument <code>upn</code> . Note that if no options specify Active Directory computer object creation and no Active Directory computer object already exists, <code>precreate_computer</code> will fail.
-all	Creates an Active Directory computer object (if one doesn't exist already), a service



Option	Description
	connection point for the computer object, and a computer zone for the computer object: in essence all of the previous three options combined.
-container	Stores the new Active Directory computer object (if created) in the Active Directory container specified by <code>rdn</code> , which is the relative distinguished name (RDN) of the container. The root of the specified Active Directory container is the distinguished name (DN) of the current domain. <code>precreate_computer</code> appends the RDN to the root DN to come up with the container DN.
-czone	Creates a computer zone for the computer object.
-dnsname	Sets the DNS name for the computer account to the provided DNS name. If this option isn't present, the <code>precreate_computer</code> command automatically sets the DNS name for the computer account. It derives the DNS name from the computer's <code>sAMAccountName</code> and the domain name.
-enctype	Set the <code>msDS-EncryptionTypes</code> permitted in <code>precreate_computer</code> command. Default is 31. Options are: <ul style="list-style-type: none"><li>■ <code>aes256-cts-hmac-sha1-96</code>, <code>aes256-cts</code></li><li>■ <code>aes128-cts-hmac-sha1-96</code>, <code>aes128-cts</code></li><li>■ <code>arcfour-hmac</code>, <code>rc4-hmac</code>, <code>arcfour-hmac-md5</code></li><li>■ <code>des-cbc-md5</code>, <code>des</code></li><li>■ <code>des-cbc-crc</code></li></ul>
-licensetype	Specifies the type of license a computer uses. The valid values are <ul style="list-style-type: none"><li>■ <code>server</code></li><li>■ <code>workstation</code></li></ul>
-nogc	Allows you to create the computer account without binding to a global catalog domain controller. You should only use this option if you know the computer <code>SCP</code> object does not exist in the domain.
-scp	Creates a service connection point for the Active Directory computer object.
-stype	Specifies the service principal types to create for a precreated computer account. You can specify multiple <code>-stype</code> options, with each specifying a different service principal type. If you don't specify this option, the <code>precreate_computer</code> command automatically creates the several default service principal names for the following service principal types: <ul style="list-style-type: none"><li>■ <code>ipp</code></li><li>■ <code>afpserver</code></li></ul>

Option	Description
	<ul style="list-style-type: none"> <li>■ nfs</li> <li>■ cifs</li> <li>■ ftp</li> <li>■ http</li> <li>■ host</li> </ul> <p>For each type of service, the <code>precreate_computer</code> command specifies two service principal names in the form of <code>serviceName/computerName</code> and <code>serviceName/computerName.domain.com</code>. For example:</p> <pre>ftp/rhe16 ftp/rhe16.acme.com</pre> <p>If you specify one or more <code>-stype</code> options, only the service principal names for those service types are created for the precreated computer account.</p>
<code>-trustee</code>	<p>Gives the user or group specified by the <code>upn</code> argument permission to join a computer to the precreated computer account. You can specify multiple <code>-trustee</code> options, with each specifying a different user or group, to give multiple users and groups permission to join a precreated computer to a zone.</p>

## Arguments

This command takes the following argument:

Argument	Type	Description
<code>samaccount@domain</code>	string	<p>Required. Specifies the name of the computer account and the domain to join. The computer name is the <code>SAMAccountName</code> for the account in the form of <code>computer\$</code>.</p> <p>For example:</p> <pre>engserv\$@acme.com</pre>

## Return value

This command returns nothing if it runs successfully.



## Examples

```
precreate_computer redhat$@acme.com -trustee  
adam.avery@acme.com  
-trustee martin.moore@acme.com -enctype arcfour-hmac
```

This example precreates a zone profile in the currently selected zone for the computer “redhat\$@acme.com”, and specifies as trustees the Active Directory users Adam Avery and Martin Moore.

Because the example does not include the `-stype` option, this example also automatically creates the following default service principal names for services on the computer:

- `ipp/redhat` and `ipp/redhat.acme.com`
- `afpserver/redhat` and `afpserver/redhat.acme.com`
- `nfs/redhat` and `nfs/redhat.acme.com`
- `cifs/redhat` and `cifs/redhat.acme.com`
- `ftp/redhat` and `ftp/redhat.acme.com`
- `http/redhat` and `http/redhat.acme.com`
- `host/redhat` and `host/redhat.acme.com`

## Related Tcl library commands

The following commands perform actions related to this command:

- `list_zones` returns a list of zones in a specified domain to `stdout`.
- `create_assignment` creates a new role assignment and saves it to Active Directory.

## `remove_user_from_group`

Use the `remove_user_from_group` command to remove an Active Directory user from an Active Directory group.

.....

## Syntax

```
remove_user_from_group user group
```

## Options

This command takes no options.

## Arguments

This command takes the following arguments:

Argument	Type	Description
user	string	Required. The user principal name (UPN) of the Active Directory user to remove.
group	string	Required. The UPN of the Active Directory group from which to remove the user.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
remove_user_from_group adam.avery@acme.com pubs@acme.com
```

## Related Tcl library commands

The following commands perform actions related to this command:

- `create_aduser` creates a new Active Directory user account and sets its password.



- `create_adgroup` creates a new Active Directory group account and specifies its scope.
- `create_user` creates a new zone user and user profile based on an existing Active Directory user.
- `create_group` creates a new zone group and group profile based on an existing Active Directory group.
- `add_user_to_group` adds an Active Directory user to an Active Directory group.

## set\_change\_pwd\_allowed

Use the `set_change_pwd_allowed` command to modify the `ADS_UF_PASSWD_CANT_CHANGE` flag in the `UserAccountControl` attribute. This flag controls whether an Active Directory user can change his or her domain password. You must specify the distinguished name of a valid Active Directory user account that should be allowed to change his or her password.

### Syntax

```
set_change_pwd_allowed userdn
```

### Options

This command takes no options.

### Arguments

This command takes the following argument:

Argument	Type	Description
userdn	string	Required. Specifies the distinguished name of the Active Directory user who is allowed to change his or her password.

.....

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_change_pwd_allowed
CN=frank.smith,CN=Users,DC=ajax,DC=test

get_object_field sd

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)
(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;PS)
```

This example deselects the “User cannot change password” account property for the Active Directory user `frank.smith`.

## Related Tcl library commands

The following commands perform actions related to this command:

- `create_aduser` creates a new Active Directory user account and sets the password for the account.
- `set_change_pwd_denied` prevents an Active Directory user from changing the domain password for his or her account.

## set\_change\_pwd\_denied

Use the `set_change_pwd_denied` command to modify the `ADS_UF_PASSWD_CANT_CHANGE` flag in the `UserAccountControl` attribute. This flag controls whether an Active Directory user can change his or her domain password. You must specify the distinguished name of a valid Active Directory user account that should not be allowed to change his or her password.

## Syntax

```
set_change_pwd_denied userdn
```

.....

## Options

This command takes no options.

## Arguments

This command takes the following argument:

Argument	Type	Description
userdn	string	Required. Specifies the distinguished name of the Active Directory user who is not allowed to change his or her password.

## Return value

This command returns nothing if it runs successfully.

## Examples

```
set_change_pwd_denied  
CN=adam.avery,CN=Users,DC=ajax,DC=test  
  
get_object_field sd  
  
(OD;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)  
(OD;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;PS)
```

This example selects the “User cannot change password” account property for the Active Directory user adam.avery.

## Related Tcl library commands

The following commands perform actions related to this command:

- [create\\_aduser](#) creates a new Active Directory user account and sets the password for the account.
- [set\\_change\\_pwd\\_allowed](#) allows an Active Directory user to change the domain password for his or her account.

# Timebox value format

A Centrify role specifies a collection of rights. A role object contains a field, `timebox`, that defines what hours in a week 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 a role's `timebox` field using the ADEdit command `get_role_field` and set the `timebox` value using `set_role_field`. You can modify an existing `timebox` value one hour at a time using the ADEdit library command `modify_timebox`.

To interpret a `timebox` value, or to set it directly, you must know the `timebox` value format which is, unfortunately, not simple as defined by Active Directory. This appendix explains the format.

## Hex string

The `timebox` value is a 42-character (21-byte) hexadecimal value stored as a string. When the hex value is converted to a binary value, its 168 bits each map to a single hour within the week. If a bit is set to 1, its corresponding hour is enabled for the role. If set to 0, its corresponding hour is disabled.

After you define the 168 bits using a hexadecimal value, you can use the `encode_timebox` function to convert the value into an internal format that specifies when a role is available to use.

## 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.



## Byte 0

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)

## Byte 1

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)

## Byte 2

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



Hour	Bit
10-11 PM	6
11-12 PM	7 (most-significant bit)

## Day mapping

Each of the seven days in a week have three bytes within the 21-byte timebox value. These bytes are in chronological order from most-significant byte to least-significant byte. (Note that this is the opposite of chronological bit order within each byte, which is LSB to MSB.) The starting point of a week is 4 PM on Saturday afternoon.

The table below shows how each day's three bytes (0-2) map to the timebox value's bytes, listed here in order from most-significant byte to least-significant byte.

Day byte	Timebox value byte
Saturday, byte 2	20 (most-significant byte)
Sunday, byte 0	19
Sunday, byte 1	18
Sunday, byte 2	17
Monday, byte 0	16
Monday, byte 1	15
Monday, byte 2	14
Tuesday, byte 0	13
Tuesday, byte 1	12
Tuesday, byte 2	11
Wednesday, byte 0	10
Wednesday, byte 1	9
Wednesday, byte 2	8
Thursday, byte 0	7
Thursday, byte 1	6
Thursday, byte 2	5
Friday, byte 0	4
Friday, byte 1	3



<b>Day byte</b>	<b>Timebox value byte</b>
Friday, byte 2	2
Saturday, byte 0	1
Saturday, byte 1	0 (least-significant byte)



# Using ADEdit with classic zones

Centrify supports both classic and hierarchical zones. If you have upgraded agents to a version of Centrify software that supports hierarchical zones (version 5.x or later), you can choose to either migrate your classic zones into a hierarchical zone structure or maintain them as classic zones.

If you choose to maintain any zones as classic zones, however, you should be aware that the authorization model in classic zones differs from the authorization model used in hierarchical zones. For example, in classic zones, authorization is an optional feature that can be enabled or disabled. If authorization is not enabled in a classic zone, any user with a valid profile in a zone is automatically granted login access to all computers joined to that zone.

Because authorization is handled differently in classic zones, there are specialized ADEdit commands and command options for creating and managing rights and roles in classic zones. The commands in this appendix are only applicable when you are working with classic zones.

## Enabling authorization in classic zones

The following ADEdit commands are used to enable or disable authorization in a classic zone and to check whether authorization is currently enabled or disabled.

Command	What it does
<code>is_dz_enabled</code>	Checks whether authorization is enabled in a currently selected classic zone.
<code>manage_dz</code>	Enables or disables authorization in classic zones.



## Working with privileged commands and PAM applications

With some limitations, you can use most of the AEdit commands for working with rights, role definitions, and role assignments in classic zones in the same way you work with them in hierarchical zones. In a classic zone, however, you must explicitly enable authorization for the zone. Thereafter, defining rights and roles or making role assignments work the same in classic zones and hierarchical zones.

In most cases, any differences or limitations for classic zones involve options or arguments that are not supported or not applicable in classic zones. For example, fields such as `allowLocalUser`, `alwaysPermitLogin`, and `auditLevel` are not applicable in classic zones. You can use the `set_role_field` command to set other field values in a classic zone. Individual commands specify these types of limitations.

## Working with restricted shell environments and commands

Before you can use the restricted shell (`dzsh`) to run commands in a classic zone, you must create the restricted shell environment. After you have created the restricted shell environment in your working context, you can run restricted shell commands in that `dzsh` context.

Restricted commands cannot be assigned to a role directly. A restricted shell environment has to be created first. The restricted shell commands can then be created under the currently selected restricted shell environment. Only one restricted shell environment can be assigned to a role. The restricted shell environment and privileged UNIX commands cannot be assigned to a role simultaneously. Assigning a new restricted shell environment to a role removes all of the previously defined privileged UNIX commands from the restricted shell. Assigning new privileged commands to a role that previously had a restricted shell environment removes the restricted shell environment and any restricted shell commands defined for the restricted shell environment.



## Setting up the restricted shell environment

The following ADEdit commands are used to set up and manage the restricted shell environment prior to working with any restricted shell commands.

Command	What it does
<code>clear_rs_env_from_role</code>	Removes the restricted shell environment from the currently selected role that is stored in memory.
<code>delete_rs_env</code>	Deletes the currently selected restricted environment from Active Directory and also from memory.
<code>get_role_rs_env</code>	Gets the restricted shell environment from the currently selected role that is stored in memory.
<code>get_rs_envs</code>	Gets the list of restricted environments that are defined within the currently selected zone.
<code>get_rse_cmds</code>	Gets a Tcl list of restricted shell commands associated with the currently selected restricted shell environment.
<code>get_rse_field</code>	Gets the value for a specified field from the restricted shell environment stored that is stored in memory.
<code>list_rs_envs</code>	Prints a list of restricted shell environments defined for the currently selected zone to stdout.
<code>new_rs_env</code>	Creates a new restricted shell environment for the current zone, stores it in memory, and sets it to be the currently selected restricted shell environment.
<code>save_rs_env</code>	Saves the currently selected restricted environment that is stored in memory to Active Directory.
<code>select_rs_env</code>	Retrieves a restricted shell environment for the currently selected zone from Active Directory, stores it in memory, and sets it to be the currently selected restricted shell environment for other ADEdit commands.
<code>set_rs_env_for_role</code>	Assigns a restricted shell environment to the currently selected role that is stored in memory.
<code>set_rse_field</code>	Sets the value for a specified field in the currently selected restricted shell environment stored in memory.

## Using restricted commands

The following ADEdit commands are used to set up and manage the restricted shell restricted shell commands.



Command	What it does
<code>delete_rs_command</code>	Deletes the currently selected restricted shell command from Active Directory and from memory.
<code>get_role_rs_commands</code>	Returns a Tcl list of restricted shell commands associated with the currently selected role.
<code>get_rs_commands</code>	Checks Active Directory and returns a Tcl list of restricted shell commands defined for the currently selected zone.
<code>get_rsc_field</code>	Gets the value for a specified field from the currently selected restricted shell command that is stored in memory.
<code>list_rs_commands</code>	Prints a list of restricted shell commands defined for the currently selected zone to <b>stdout</b> .
<code>new_rs_command</code>	Creates a new restricted shell command under the currently selected restricted shell environment, stores it in memory, and sets it to be the currently selected restricted shell command.
<code>save_rs_command</code>	Saves the currently selected restricted shell command that is stored in memory to Active Directory.
<code>select_rs_command</code>	Retrieves a restricted shell command in the currently selected zone from Active Directory, stores it in memory, and sets it to be the currently selected restricted shell command for other ADEdit commands.
<code>set_rsc_field</code>	Sets the value for a specified field for the currently selected restricted shell command that is stored in memory.

## Creating computer-level role assignments in classic zones

Classic zones support computer-level role assignments. If you want to configure computer-level role assignments, keep the following in mind:

- The classic zone that the computer is a member of must have authorization enabled before you can create role definitions and role assignments.
- The role assignment is only valid on the computer where you have made the assignment.
- The role definition you use must be defined in the classic zone that the computer is a member of.

A computer-level role assignment in a classic zone is similar to computer-level overrides in hierarchical zones, except that you cannot save user or group profile information for individual computers. User and group information is

• • • • •

stored in the classic zone. To enable computer-specific role assignments in classic zones, you must use a specialized zone type, the `classic-computer` zone type.

## To create a computer-level role assignment in a classic zone:

1. Precreate the computer in a classic4 zone, if it doesn't already exist.
2. Create a zone that uses the specialized zone type of `classic-computer`.
3. Select the `classic-computer` zone within the classic zone.
4. Create the role assignment.

The following code snippet illustrates the commands to execute in ADEdit to make computer-specific role assignments in classic zones:

```
bind ajuba.net
package require ade_lib
    1.0
select_zone cn=cls,cn=zones,dc=ajuba,dc=net
get_zone_field type
    classic4
precreate_computer rhelqa$@ajuba.net
get_zone_computers
    {comp5$@ajuba.net} {rhelqa$@ajuba.net}
create_zone classic-computer
rhelqa.ajuba.net@cn=cls,cn=zones,dc=ajuba,dc=net
select_zone
rhelqa.ajuba.net@cn=cls,cn=zones,dc=ajuba,dc=net
new_role_assignment user5@ajuba.net
set_role_assignment_field role role1/cls
save_role_assignment
```

You can then get the `classic-computer` zones by running the `get_child_zones` command when the classic zone is selected. For example:

```
select_zone cn=cls,cn=zones,dc=ajuba,dc=net
get_child_zones
rhelqa.ajuba.net@CN=c122,CN=Zones,DC=ajuba,DC=net
comp5.ajuba.net@CN=c122,CN=Zones,DC=ajuba,DC=net
```

# Quick reference for commands and library procedures

The following table lists the ADEdit and `ade_lib` commands in alphabetical order. The table summarizes the command syntax for each command with optional elements in [square brackets] and variables in *italics*. For more detailed information about any command, see [ADEdit command reference](#) or [ADEdit Tcl procedure library reference](#)

Command syntax	Abbreviation	ade_lib
<code>add_command_to_role</code> <i>command</i> [/ <i>zonename</i> ]	acr	
<code>add_map_entry</code> <i>key value</i>	ame	
<code>add_map_entry_with_comment</code> <i>key value comment</i>	amewc	
<code>add_object_value</code> <i>dn field value</i>	aov	
<code>add_pamapp_to_role</code> <i>app</i> [/ <i>zonename</i> ]	apr	
<code>add_sd_aces</code> <i>ddl_string ace_string</i>	ase	
<code>add_user_to_group</code> <i>user group</i>		X
<code>bind</code> [-gc] [-write] [-machine] [ <i>server@</i> ] <i>domain</i> [ <i>user</i> [ <i>password</i> ]]		
<code>clear_rs_env_from_role</code>	crse	
<code>convert_msdate</code> <i>msdate</i>		X
<code>create_adgroup</code> <i>dn sam gtype</i>		X
<code>create_aduser</code> <i>dn upn sam pw</i>		X
<code>create_assignment</code> <i>upn role</i> [/ <i>zonename</i> ] [ <i>from</i> ] [ <i>to</i> ] [ <i>description</i> ]		X
<code>create_computer_role</code> <i>computer_role_path group_upn</i>	ccr	
<code>create_dz_command</code> <i>name command description form dzdo_runas dzsh_runas flags pri umask path</i>		X



Command syntax	Abbreviation	ade_ lib
<code>create_group</code> <i>upn name gid</i>		X
<code>create_nismap</code> <i>map key:value comment</i>		X
<code>create_pam_app</code> <i>name application description</i>		X
<code>create_role</code> <i>name description sysrights pamrights cmdrights allowlocal rsenv visible</i>		X
<code>create_rs_command</code> <i>rsc_name cmd description form dzsh_runas flags pri umask path</i>		X
<code>create_rs_env</code> <i>rse_name rse_description</i>		X
<code>create_userad</code> <i>uname uid gid gecos home shell role</i>		X
<code>create_zone</code> [-ou] <i>zone_type path [schema_type]</i>	cz	
<code>decode_timebox</code> <i>strTimeBox</i>		X
<code>delegate_zone_right</code> <i>right principal_upn</i>		
<code>delete_dz_command</code>	dldzc	
<code>delete_local_group_profile</code> <i>group_name</i>	dllgp	
<code>delete_local_user_profile</code> <i>user_name</i>	dllup	
<code>delete_map_entry</code> <i>key: index</i>	dlme	
<code>delete_nis_map</code>	dlnm	
<code>delete_object</code>	dlo	
<code>delete_pam_app</code>	dlpam	
<code>delete_role</code>	dlr	
<code>delete_role_assignment</code>	dlra	
<code>delete_rs_command</code>	dlrsc	
<code>delete_rs_env</code>	dlrse	
<code>delete_sub_tree</code> <i>dn</i>		
<code>delete_zone</code>	dlz	
<code>delete_zone_computer</code>	dlzc	
<code>delete_zone_group</code>	dlzg	
<code>delete_zone_user</code>	dlzu	
<code>dn_from_domain</code> <i>domain_name</i>	dnfd	
<code>dn_to_principal</code> [-upn] <i>principal_dn</i>	dntp	
<code>domain_from_dn</code> <i>domain_name</i>	dfdn	
<code>encode_timebox</code> <i>strTimeBox</i>		X
<code>explain_group</code> <i>Type gt</i>		X



Command syntax	Abbreviation	ade_ lib
<code>explain_ptypept</code>		X
<code>explain_sdsddl_string</code>		
<code>explain_trustAttributesta</code>		X
<code>explain_trustDirection td</code>		X
<code>explain_userAccountControl uac</code>		X
<code>get_adinfo domain zone host</code>	adinfo	
<code>get_all_zone_users [-upn] zone_DN</code>		X
<code>get_bind_info domain forest server sid domain_level forest_level</code>	gbi	
<code>get_child_zones [-tree] [-crole] [-computer]</code>	gcz	
<code>get_dz_commands</code>	gdzc	
<code>get_dzc_field field</code>	gdzcf	
<code>get_effective_groups [-hostname computer_name]</code>		X
<code>get_effective_users [-hostname computer_name]</code>		X
<code>get_group_members [-ad   -upn] group_UPN</code>	ggm	
<code>get_local_group_profile_field field_name</code>	glgpf	
<code>get_local_groups_profile</code>	glgp	
<code>get_local_user_profile_field field_name</code>	glupf	
<code>get_local_users_profile</code>	glup	
<code>get_nis_map</code>	gnm	
<code>get_nis_map_field field</code>	gnmf	
<code>get_nis_map_with_comment</code>	gnmwc	
<code>get_nis_maps</code>	gnms	
<code>get_object_field field</code>	gof	
<code>get_object_field_names</code>	gofn	
<code>get_objects [-gc] [-depth one sub] [-limit limit] [-f forest] base filter</code>	go	
<code>get_pam_apps</code>	gpam	
<code>get_pam_field</code>	gpf	
<code>get_parent_dn DN</code>	gpd	
<code>get_pending_zone_groups</code>	gpzg	
<code>get_pending_zone_users</code>	gpzu	
<code>get_pwnam_unix_name</code>	gpn	
<code>get_rdn DN</code>	grdn	



Command syntax	Abbreviation	ade_ lib
get_role_apps	grap	
get_role_assignment_field <i>field</i>	graf	
get_role_assignments [-upn]	gra	
get_role_commands	grc	
get_role_field <i>field</i>	grf	
get_role_rs_commands	grrsc	
get_role_rs_env	grrse	
get_roles	getr	
get_rs_commands	grsc	
get_rs_envs	grse	
get_rsc_field <i>field</i>	grscf	
get_rse_cmds	grsec	
get_rse_field <i>field</i>	grsef	
get_effective_groups [-dn] [-z] <i>user_DN user_UPN</i>		X
get_user_role_assignments [-visible] [-hostname hostname] <i>user_DN</i>		X
get_schema_guid <i>schema_name</i>	gsg	
get_zone_computer_field <i>field</i>	gzcf	
get_zone_computers	gzc	
get_zone_field <i>field</i>	gzf	
get_zone_group_field <i>field</i>	gzgf	
get_zone_groups	gzg	
get_zone_nss_vars	gznv	
get_zone_user_field <i>field</i>	gzuf	
get_zone_users [-upn]	gzu	
get_zones <i>domain</i>	gz	
getent_passwd	gep	
guid_to_id <i>guid</i>		
help <i>command_pattern</i>	h	
is_dz_enabled	idze	
joined_get_user_membership <i>user_UPN</i>	jgum	
joined_name_to_principal [-upn] <i>UNIX_name</i>	jntp	
joined_user_in_group <i>user_UPN group_UPN</i>	jug	



Command syntax	Abbreviation	ade_ lib
list_dz_commands	lsdzc	
list_local_groups_profile	lslgp	
list_local_users_profile	lslup	
list_nis_map	lsnm	
list_nis_map_with_comment	lsnmwc	
list_nis_maps	lsnms	
list_pam_apps	lspa	
list_pending_zone_groups	lpzg	
list_pending_zone_users	lpzu	
list_role_assignments [-upn] [-visible] [-user   -group   -invalid]	lsra	
list_role_rights	lsrr	
list_roles	lsr	
list_rs_commands	lsrsc	
list_rs_envs	lsrse	
list_zone_computers	lszc	
list_zone_groups	lszg	
list_zone_users [-upn]	lszu	
list_zonesdomain		X
lmerge [list] [list] [list...]		X
manage_dz -on   -off	mnz	
modify_timebox strTimeBox day hour avail		X
move_objectdestinationDN	mvo	
new_dz_commandname	newdzc	
new_local_group_profilegroup_name	newlgp	
new_local_user_profileuser_name	newlup	
new_nis_map [-automount] map	newnm	
new_objectdn	newo	
new_pam_appname	newpam	
new_rolename	newr	
new_role_assignmentupn	newra	
new_rs_commandname	newrsc	
new_rs_envname	newrse	



Command syntax	Abbreviation	ade_ lib
<code>new_zone_computer</code> <i>sAMAccountName@domain</i>	newzc	
<code>new_zone_group</code> <i>AD_group_UPN</i>	newzg	
<code>new_zone_user</code> <i>AD_user_UPN</i>	newzu	
<code>pop</code>		
<code>precreate_computer</code> <i>AMAccount@domain</i> [-ad] [-scp] [-czone] [-all] [-container <i>rdn</i> ] [-dnsnamednsname] [-trustee <i>upn</i> [-trustee <i>upn</i> ] ...] [-nogc] [-stype <i>spn</i> [-stype <i>spn</i> ] ...]		X
<code>principal_from_sid</code> [-upn] <i>sid</i>	pfs	
<code>principal_to_dn</code> <i>principal_upn</i>	ptd	
<code>principal_to_id</code> [-apple] <i>upn</i>	pti	
<code>push</code>		
<code>quit</code>	q	
<code>remove_command_from_role</code> <i>command</i> [/ <i>zonename</i> ]	rcfr	
<code>remove_object_value</code> <i>dn field value</i>	rov	
<code>remove_pamapp_from_role</code> <i>app</i> [/ <i>zonename</i> ]	rpamfr	
<code>remove_sd_aces</code> <i>ddl_string ace_string</i>	rsa	
<code>remove_user_from_group</code> <i>user group</i>		X
<code>rename_object</code> <i>name</i>	rno	
<code>save_dz_command</code>	svdzc	
<code>save_local_group_profile</code>	svlgp	
<code>save_local_user_profile</code>	svlup	
<code>save_nis_map</code>	svnm	
<code>save_object</code>	svo	
<code>save_pam_app</code>	svpam	
<code>save_role</code>	svr	
<code>save_role_assignment</code>	svra	
<code>save_rs_command</code>	svrsc	
<code>save_rs_env</code>	svrse	
<code>save_zone</code>	svz	
<code>save_zone_computer</code>	svzc	
<code>save_zone_group</code>	svzg	
<code>save_zone_user</code>	svzu	
<code>select_dz_command</code> <i>command</i>	sldzc	



Command syntax	Abbreviation	ade_ lib
<code>select_local_group_profile</code> <i>group_name</i>	slgp	
<code>select_local_user_profile</code> <i>user_name</i>	slup	
<code>select_nis_map</code> <i>map</i>	slnm	
<code>select_object</code> [-rootside] [-attrs <i>a1[,a2,...]</i> ] <i>dn</i>	slo	
<code>select_pam_app</code> <i>name</i>	slpam	
<code>select_role</code> <i>role</i>	slr	
<code>select_role_assignment</code> <i>principal/role[/zone]</i>	slra	
<code>select_rs_command</code> <i>rs_cmd</i>	slrsc	
<code>select_rs_env</code> <i>rse</i>	slrse	
<code>select_zone</code> <i>path</i>	slz	
<code>select_zone_computer</code> <i>SAMaccountName@domain</i>	slzc	
<code>select_zone_group</code> <i>AD_group_UPN</i>	slzg	
<code>select_zone_user</code> <i>user</i>	slzu	
<code>set_change_pwd_allowed</code> <i>userdn</i>		
<code>set_change_pwd_denied</code> <i>userdn</i>		
<code>set_dzc_field</code> <i>field value</i>	sdzcf	
<code>set_ldap_timeout</code> <i>timeout_in_seconds</i>		
<code>set_local_group_profile_field</code> <i>field_name value</i>	slgpf	
<code>set_local_user_profile_field</code> <i>field_name value</i>	slupf	
<code>set_object_field</code> <i>field value</i>	sof	
<code>set_pam_field</code> <i>field value</i>	spf	
<code>set_role_assignment_field</code> <i>field value</i>	sraf	
<code>set_role_field</code> <i>field value</i>	srf	
<code>set_rs_env_for_role</code> <i>environment</i>	srse	
<code>set_rsc_field</code> <i>field value</i>	srsf	
<code>set_rse_field</code> <i>field value</i>	srsef	
<code>set_sd_owners</code> <i>ddl_string owner_sid</i>	sso	
<code>set_user_password</code> <i>principal_UPN password</i>	sup	
<code>set_zone_computer_field</code> <i>field value</i>	szcf	
<code>set_zone_field</code> <i>field value</i>	szf	
<code>set_zone_group_field</code> <i>field value</i>	szgf	
<code>set_zone_user_field</code> <i>field value</i>	szuf	



Command syntax	Abbreviation	ade_ lib
<code>show</code> [all   bind   zone   user   computer   assignment   object   group   pamright   dzcommand   nismap   role   license   rse   rs_command]		
<code>sid_to_escaped_string</code> <i>sid</i>	stes	
<code>sid_to_uid</code> <i>sid</i>	stu	
<code>validate_license</code> <i>path</i>	vl	