Role-Based Access Controls in Data ONTAP™: Granular Administration of Capabilities

Christian D. Odhner | TR-3358 | October 2004

TECHNICAL REPORT

Network Appliance, a pioneer and industry leader in data storage technology, helps organizations understand and meet complex technical challenges with advanced storage solutions and global data management strategies.

Executive Summary

This paper is intended for storage administrators, security administrators, and IT management. It describes the role-based access controls (RBAC) introduced with Data ONTAP 7G, an overview of the benefits of this feature, and some deployment examples. The appendices include reference information that should be of value for storage and security administrators who perform RBAC configuration.

Table of Contents

1.	**What Are Role-Based Access Controls?**	**3**
2.	**How Does RBAC Work in Data ONTAP?**	**3**
2.1.	Users	3
2.2.	Groups	4
2.3.	Roles	4
2.4.	Capabilities	4
2.5.	Putting It All Together	4
3.	**Integration with Microsoft Active Directory**	**5**
4.	**Deployment Examples**	**5**
4.1.	CIFS File Services in a DMZ Environment	5
4.2.	Separate CIFS, NFS, and Storage Administration Groups	6
4.3.	Separate NAS (CIFS and NFS) and SAN (FCP and iSCSI) Administration Groups	7
4.4.	Performance-Monitoring Pseudouser for Scripts	7
4.5.	Snapshot Copy Creation for Database Backups	7
5.	**How RBAC Interacts with Data ONTAP Upgrades and Downgrades**	**8**
6.	**Summary**	**8**
7.	**References**	**8**
8.	**Appendices**	**8**
8.1.	Appendix A: Login Capabilities	8
8.2.	Appendix B: Security Capabilities	9
8.3.	Appendix C: Command-Line Interface (CLI) Capabilities	9
8.4.	Appendix D: Application Programming Interface (API) Capabilities	11

1. What Are Role-Based Access Controls?

Role-based access controls, or RBAC, are a method for managing the set of actions that a user or administrator may perform in a computing environment.

Historically, older computer operating systems allowed any user who had access to the system to perform any function. In fact, many systems did not distinguish between users at all. Most current operating systems provide, at a minimum, the ability to create several different users, each with a separate username and password. Once the ability to distinguish between users was provided, operating systems began to use user identification as a means to control access to files, directories, and other system objects. Good examples of this are the file permissions used on UNIX® systems (and the NFS protocol) or the access control lists (ACLs) used on Windows® systems (and the CIFS protocol).

In addition to file access, there are other actions that should be managed for security reasons. For example, only the system administrator should be allowed to add new user accounts to the system. From this it becomes clear that the users who access a system fall into at least two categories, or roles: administrators and nonadministrators.

While reserving certain functions for administrator-only access is a good start, additional problems need to be solved. Most organizations have multiple system administrators, some of whom require more privileges than others. By selectively granting or revoking privileges for each user, you can customize the degree of access that an administrator has to the system. As one example, Microsoft® Windows provides this capability. The problem with this approach is that as the number of system administrators grows, it becomes difficult and time consuming to manage the set of capabilities granted to each administrator.

Role-based access controls solve this management problem by allowing you to define sets of capabilities (*roles*) that are not assigned to any particular user. Users are assigned to groups based on their job functions, and each group is granted the set of roles required to perform those functions. Using this method, the only configuration required for an individual administrator is to ensure that that administrator is a member of the appropriate groups; that administrator will inherit all the correct capabilities because of the group membership and the roles assigned to those groups.

2. How Does RBAC Work in Data ONTAP?

While the overall concept of role-based access controls is applicable to a wide range of operating systems and applications, the details of how RBAC is implemented vary depending on the OS or application in use. This section describes the specific terminology and architecture used in Data ONTAP; it is important to understand these concepts and definitions before configuring RBAC in Data ONTAP, especially if you have experience with RBAC implementations in other software (because the terminology or architecture might be different from implementations you have used in the past).

2.1. Users

A *user* is defined as an account that is authenticated on the NetApp system.

A *domain user* is defined as a nonlocal user who belongs to a Windows domain and is authenticated by the domain.

Both users and domain users represent individual, authenticated humans. While it is possible to define a user or domain user that represents a piece of software or that is shared among multiple humans (an example of this usage is provided in section 4), this is not the most common scenario and is not discussed in depth here.

Both users and domain users, as discussed in this document, are assumed to be authorized system administrators. Normal, nonadministrative users who access files on the system via CIFS or NFS, or who use client systems that mount LUNs via FCP or iSCSI, are not discussed in this document. They have no ability to log into or manage a Data ONTAP system unless they have been specifically defined as either users or domain users via the useradmin command, as discussed below.

2.2. Groups

A *group* is defined as a collection of users and/or domain users. Groups may be assigned one or more roles. It is important to remember that the groups defined within Data ONTAP are separate from the groups defined in other contexts, such as a Microsoft Active Directory server. This is true even if the groups within Data ONTAP have the same names as groups elsewhere within your environment.

When creating new users and/or domain users, Data ONTAP *requires* specification of group membership. Therefore, it is best to create appropriate groups before defining users or domain users.

2.3. Roles

A *role* is defined as a named set of capabilities. Data ONTAP comes with several roles predefined, and users may create additional roles or modify the provided roles.

When creating new groups, Data ONTAP *requires* specification of the roles the new groups should have. Therefore, it is best to create appropriate roles before defining groups.

2.4. Capabilities

A *capability* is defined as the privilege granted to a role to execute commands or take other specified actions.

Data ONTAP uses four different types of capabilities:

- **Login rights.** These capabilities have names that begin with “login-“ and are used to control which access methods an administrator is permitted to use for managing the system.
- **CLI rights.** These capabilities have names that begin with “cli-“ and are used to control which commands the administrator may use within the Data ONTAP command-line interface.
- **API rights.** These capabilities have names that begin with “api-“ and are used to control which application programming interface (API) commands may be used. API commands are usually executed by programs, rather than directly by administrators; however, you might wish to restrict a specific program to certain APIs by creating a special user account for it, or you might wish to have a program authenticate as the administrator who is using the program and be limited by that administrator’s roles.
- **Security rights.** These capabilities have names that begin with “security-“ and are used to control the ability to use advanced commands or to change passwords for other users.

2.5. Putting It All Together

Users are members of groups, groups have one or more roles, and each role grants a set of capabilities. In this way Data ONTAP allows you to create flexible security policies that match your organizational needs.

All configuration for role-based access controls occurs via the *useradmin* command provided by Data ONTAP. For example, users are added or modified with the useradmin user add or useradmin user modify commands. This section includes specific command-line instructions from the *Data ONTAP System Administration Guide* for your convenience. Example deployment scenarios are provided in section 4 of this paper.

Because users and domain users must be members of groups, and because groups must be assigned one or more roles, the best sequence for configuration tasks is to create the roles first; then create groups and assign roles to them; and finally create users and domain users, providing them with appropriate group membership.

Detailed documentation on how to use the *useradmin* command to define users, domain users, groups, and roles is provided in the *Data ONTAP System Administration Guide*. This paper is not intended to replace the documentation in the *System Administration Guide*.

3. Integration with Microsoft Active Directory

The ability to define domain users that are authenticated by an Active Directory domain rather than Data ONTAP is a powerful tool for managing large storage environments. Most enterprise computing environments already have an Active Directory infrastructure available, and storage administrators or other users who need administrative access to storage devices already have accounts defined within that Active Directory infrastructure. By using this preexisting authentication capability, rather than defining separate accounts for the storage environment, several key benefits are obtained:

- An administrator's authentication credentials (username, password) are the same when logging into the storage system as they are when logging into any Windows system in the environment. When the password is changed within the Windows environment, the change immediately takes effect within the storage environment.
- Additionally, changing an administrator's password once, within Active Directory, has the effect of changing it on all storage devices to which that administrator has access. This is a significant reduction in management overhead for environments with a large number of storage devices.
- Centralized authentication allows local security policy, implemented within Active Directory, to take effect across all storage devices as well. For example, administrators may be compelled to change their passwords with a certain frequency and may be provided with advance warning as password expiration time approaches. Likewise, when they do change passwords, the Active Directory environment can enforce policy regarding password composition and length checking, reuse of previous passwords, use of dictionary words in passwords, etc.
- When an administrator departs from an organization, disabling that administrator's Active Directory account has the side effect of immediately revoking access to the storage environment as well.

However, it would be inadvisable to provide *all* of the accounts within Active Directory access to storage management functions. Obviously only a subset of the AD accounts represent administrative staff, and only a subset of the administrative staff (in any large organization) will have a need to administer filer or NearStore® systems. Any system that provides transparent Active Directory authentication on a storage system without discriminating between authorized administrators and other accounts will be exposing the storage system to huge security problems.

To avoid such problems, Data ONTAP only authenticates an administrator against Active Directory if that administrator has been defined as a domain user using the useradmin command.

To maximize the benefit of this new capability, consider migrating filer and NearStore administrator accounts from local users to domain users. This is a simple process:

1. Ensure that the storage devices have been upgraded to Data ONTAP 7G or later.
2. Use the *useradmin userdel* command to delete the local usernames and passwords.
3. Use the *useradmin domainuser add* command to grant authorized Active Directory accounts administrator access.

4. Deployment Examples

Data ONTAP role-based access controls have the flexibility to meet the needs of almost any IT environment. How they are used will depend largely on local security policies and organizational structure. The following are just a few examples of how RBAC might be used to enhance security and manageability in an enterprise IT environment.

4.1. CIFS File Services in a DMZ Environment

Some DMZ environments need to provide CIFS file access services without the benefit of a Windows domain controller or Active Directory server. Using RBAC, customers can set up local user accounts on the storage system for access via CIFS and configure roles and group membership to restrict administrative login to authorized administrative users.

The first step is to create a role for users who will have only CIFS access to the filer or NearStore system. We will give this role no capabilities at all, because these users require no administrative access. It is also a good idea to provide a comment that explains the purpose of the role.

Fortunately, Data ONTAP already has a default role called “none” that fills this need.

The second step is to create a group of which the CIFS-only users will be members. Again, it is a good idea to use a descriptive name and to provide a comment. For this particular purpose, several of the default groups provided by Data ONTAP (such as Everyone and Guest) have appropriate roles, but it may be a good idea to use a more descriptive name:

```
filer# useradmin group add cifsonly –c “This group is for non-admin CIFS access” –r none
```

Finally, create the individual user accounts and make them members of the group:

```
filer# useradmin user add joesmith –c “Non-administrative CIFS-only user” –n “Joe Smith” –g cifsonly
filer# useradmin user add janedoe –c “Non-administrative CIFS-only user” –n “Jane Doe” –g cifsonly
```

After each of these commands, you will be prompted to set a password for the user. Provided that the filer or NearStore system has been configured to use CIFS in workgroup mode with local authentication, these users will now have the capability to access CIFS shares (limited, of course, by the share-level and per-file access control lists) but will not be able to log into the filer for administrative access because they lack any login capabilities.

4.2. Separate CIFS, NFS, and Storage Administration Groups

Some IT organizations have separate groups of administrators for Windows, UNIX, and storage administration. Using RBAC, it is possible to provide the Windows administrators with access to CIFS configuration functions; provide the UNIX administrators with NFS configuration functions; and provide the storage administrators with volume, qtree, and other storage management functions. This prevents the Windows and UNIX administrators from mistakenly changing the configuration for protocols that are managed by the other group and allows the storage administrators to focus on allocating storage resources while delegating tasks specific to Windows and UNIX to the appropriate server administrators.

First, we need to define the roles. There are some functions that all of the administrators should have access to, such as the login capabilities and certain reporting commands (for example, df). Rather than create a role for each of these groups and duplicate the common capabilities across all three groups, we will put the common capabilities in a role that will be assigned to all three groups. Then we can give each group a second, unique role that will have the capabilities specific to that group:

```
filer# useradmin role add common –c “Capabilities common to all administrators” –a \
        login-telnet,login-ssh,cli-df*,cli-help*,cli-man*,cli-sysstat*,cli-uptime*,cli-options*,cli-passwd*
filer# useradmin role add cifsadmin –c “CIFS administrators” –a cli-cifs*,cli-nbtstat*,cli-wcc*,cli-vscan*
filer# useradmin role add nfsadmin –c “NFS administrators” –a cli-nfs*,cli-exportfs*,cli-nis*,cli-nfsstat*
filer# useradmin role add storageadmin –c “Storage administrators” –a cli-*,security-*,api-*,\
        login-console,login-http-admin
```

Note that the storage administrators are granted all of the CLI, security, and API capabilities, as well as two additonal login methods not granted to the other groups. It is certainly possible to provide a more restrictive set of capabilities to the storage administrators, but remember that a storage administrator will only be able to delegate roles and capabilities that that administrator possesses.

Once the roles are created, groups should be created that have the roles assigned to them:

```
filer# useradmin group add winadmins –c “Windows Administrators” –r common,cifsadmin
filer# useradmin group add unixadmins –c “Unix[BEZM2] Administrators” –r common,nfsadmin
filer# useradmin group add fileradmins –c “Storage Administrators” –r common,storageadmin
```

Note that the *common* role and the *storageadmin* role have some overlap in terms of CLI capabilities; however, the storage administrator group still needs the *common* role if the administrators are to have access to the login-telnet and login-ssh capabilities.

Finally, users need to be created for individual administrators. In order to provide centralized password management, we will use domainusers instead of local users:

```
filer# useradmin domainuser add bob –g fileradmins
```

```
filer# useradmin domainuser add alice –g fileradmins
filer# useradmin domainuser add johndoe –g winadmins
filer# useradmin domainuser add janedoe –g unixadmins
```

Note that the usernames above (bob, alice, johndoe, janedoe) must be valid accounts within the Windows domain of which the filer or NearStore system is a member.

4.3. Separate NAS (CIFS and NFS) and SAN (FCP and iSCSI) Administration Groups

Other IT organizations may wish to delegate responsibility based on data class, rather than client operating system types as discussed above. For example, RBAC could be used to delegate network-attached storage (NAS) protocol administration (both NFS and CIFS) to one set of administrators and storage area network (SAN) protocol administration (both FCP and iSCSI) to another set of administrators. Similarly, an IT organization might wish to set up a network administration group that has access to manage DNS, network interface addresses, and other such details, but without any protocol-level capabilities.

We can build a NAS administration group using the roles defined in the previous example (section 4.2), like this:

```
filer# useradmin group add nasadmins –c “NAS Administrators” –r common,cifsadmin,nfsadmin
```

For the SAN administrators and network administrators, new roles would be required before creating the appropriate groups:

```
filer# useradmin role add sanadmin –c “FCP and iSCSI commands” –a cli-lun*,cli-fcp*,cli-igroup*\
        cli-reallocate*,cli-iscsi*
filer# useradmin role add netadmin –c “Network commands” –a cli-ifconfig*,cli-ifstat*,cli-netstat*,\
        cli-arp*,cli-ping*,cli-traceroute*,cli-dns*,cli-route*,cli-routed*

filer# useradmin group add sanadmins –c “SAN Administrators” –r common,sanadmin
filer# useradmin group add netadmins –c “Network Administrators” –r common,netadmin
```

Users or domain users could then be added to these groups as in the above examples.

4.4. Performance-Monitoring Pseudouser for Scripts

In some environments, it is desirable to run custom scripts on some (UNIX or Windows) administration host to periodically log into the storage system and collect performance or usage data. RBAC may be used to create a special user account for this purpose, to ensure that the performance collection script has only the access it needs to obtain the required data. In this way, it is possible to prevent errors in the data collection script from executing possibly harmful commands.

While the above examples illustrate the ability to create modular, reusable roles that are combined to create appropriate groups, sometimes it is more desirable to have a one-to-one mapping between a special-purpose role and a group, as shown here:

```
filer# useradmin role add monitor –c “Performance monitoring” –a login-rsh,cli-df*,cli-sysstat*,cli-stats*
filer# useradmin group add monitor –c “Performance monitoring” –r monitor
filer# useradmin user add monitor –c “Performance monitoring” –g monitor
```

You would then be prompted to set a password for the monitor user. This user may log in, but only via RSH, and may execute only performance-monitoring commands. Of course, one could adjust this profile to use SSH instead of RSH, to add other performance-related commands (such as netstat), and so on.

4.5. Snapshot™ Copy Creation for Database Backups

Database environments frequently have a need to coordinate database activities with Snapshot copy creation on the storage system. For example, during a backup operation, the database must usually be placed into a “hot backup” or “write suspend” mode; then a Snapshot copy must be created; then the database must be placed back into normal operation. This process is usually managed from the database host and is usually performed by a database administrator (DBA) or by a script that the DBA writes and maintains. RBAC allows the storage

administrator to grant the DBA enough access to log in and create Snapshot copies as required by this process, while preventing the DBA from having access to other functions such as network configuration, security settings, or storage allocation.

For automation purposes such as this, even the *common* role defined above has capabilities that are not required. A new role, restricted to login and Snapshot management, would be better:

```
filer# useradmin role add snapstuff –c “Login and snapshot management only” –a login-ssh,cli-snap*
filer# useradmin group add snapadmin –c “Database Snapshot Management” –r snapstuff
filer# useradmin user add dba –c “Database Administrator Login” –g snapadmin
```

Of course, it would also be reasonable to set this up with a domain user for centralized password management.

5. How RBAC Interacts with Data ONTAP Upgrades and Downgrades

When you upgrade to Data ONTAP 7G or later, any existing local users are granted full privileges. To manage or reduce those privileges, you must create one or more groups with defined roles and capabilities, then place the users in groups.

If you create user accounts in Data ONTAP 7G with groups, roles, and capabilities, their specific privilege levels will not be preserved if you revert to an earlier release. The user accounts and names will be preserved during the revert, but each account will have full privileges in the earlier release.

Since domain user functionality is not *available* in releases prior to Data ONTAP 7G, any domain users should be deleted using the *useradmin domainuser delete* command prior to revert.

6. Summary

Role-based access controls in Data ONTAP allow storage and security administrators to map administrative capabilities to local security policy. This can assist in meeting regulatory or internal policy requirements, can protect in many cases against accidental misconfigurations and other user errors, and can allow storage administrators to delegate responsibility for certain tasks without providing untrained administrators with access to destructive commands.

7. References

Detailed information about managing users, domain users, groups, and roles may be found in the *Data ONTAP System Administration Guide* in Chapter 6: Managing Administrator Access. Detailed command descriptions and syntax may be found in the *Data ONTAP Commands: Manual Page Reference* document. These documents are available at *http://now.netapp.com/* under “Access Product Documentation” from the Service and Support home page. Always refer to the documentation for the version of Data ONTAP you are actually using, as some commands and features vary from release to release.

The following appendices provide a list of all of the capabilities that may be assigned to roles within Data ONTAP.

8. Appendices

8.1. Appendix A: Login Capabilities

- login-telnet
- login-console
- login-rsh
- login-ssh
- login-http-admin

Note: Administrators require at least one login capability (telnet, console, rsh, or ssh) in order to execute any CLI commands. Administrators require the login-http-admin capability in order to execute any API commands. Any administrator provided with login-http-admin is also allowed to log in via FilerView® and/or secure FilerView, which provide access to all administrative functions regardless of the user's role. For this reason only completely trusted administrators should be given access to login-http-admin.

8.2. Appendix B: Security Capabilities

- security-passwd-change-others
- security-priv-set-advanced

8.3. Appendix C: Command-Line Interface (CLI) Capabilities

- cli-adconfig*
- cli-adinfo*
- cli-adstat*
- cli-aggr*
- cli-arp*
- cli-atm*
- cli-atmarp*
- cli-atmconfig*
- cli-atmfmbstat*
- cli-backup*
- cli-bootfs*
- cli-cf*
- cli-cifs*
- cli-config*
- cli-dafs*
- cli-date*
- cli-dd*
- cli-df*
- cli-disk*
- cli-diskcopy*
- cli-disktest*
- cli-dlm*
- cli-dns*
- cli-download*
- cli-dump*
- cli-echo*
- cli-elarp*
- cli-elconfig*
- cli-ems*
- cli-enable*
- cli-environ*
- cli-environment*
- cli-exportfs*
- cli-fcdiag*
- cli-fcp*
- cli-fcstat*
- cli-fctest*
- cli-file*
- cli-filestats*
- cli-floppyboot*
- cli-fpolicy*
- cli-ftp*
- cli-ftpd*
- cli-halt*

- cli-help*
- cli-hostname*
- cli-httpstat*
- cli-ifconfig*
- cli-ifinfo*
- cli-ifstat*
- cli-igroup*
- cli-ipsec*
- cli-ipspace*
- cli-iscsi*
- cli-iswt*
- cli-license*
- cli-lock*
- cli-logger*
- cli-logout*
- cli-lun*
- cli-man*
- cli-maxfiles*
- cli-mt*
- cli-nbtstat*
- cli-ndmpcopy*
- cli-ndmpd*
- cli-netdiag*
- cli-netstat*
- cli-nfs*
- cli-nfsstat*
- cli-nis*
- cli-options*
- cli-orouted*
- cli-partner*
- cli-passwd*
- cli-ping*
- cli-priv*
- cli-qtree*
- cli-quota*
- cli-rdate*
- cli-reallocate*
- cli-reboot*
- cli-restore*
- cli-rmc*
- cli-route*
- cli-routed*
- cli-san*
- cli-savecore*
- cli-secureadmin*
- cli-setup*
- cli-shelfchk*
- cli-snap*
- cli-snapmirror*
- cli-snapvault*
- cli-snmp*
- cli-software*
- cli-source*
- cli-stats*
- cli-storage*
- cli-sysconfig*
- cli-sysstat*
- cli-timezone*
- cli-traceroute*

- cli-uniconfig*
- cli-ups*
- cli-uptime*
- cli-useradmin*
- cli-version*
- cli-vfiler*
- cli-vif*
- cli-vlan*
- cli-vol*
- cli-vscan*
- cli-wcc*
- cli-ypcat*
- cli-ypgroup*
- cli-ypmatch*
- cli-ypwhich*

8.4. Appendix D: Application Programming Interface (API) Capabilities

- api-aggr-*
- api-aggr-add
- api-aggr-create
- api-aggr-destroy
- api-aggr-get-filer-info
- api-aggr-get-root-name
- api-aggr-list-info
- api-aggr-mediascrub-list-info
- api-aggr-mirror
- api-aggr-offline
- api-aggr-online
- api-aggr-options-list-info
- api-aggr-rename
- api-aggr-restrict
- api-aggr-scrub-list-info
- api-aggr-scrub-resume
- api-aggr-scrub-start
- api-aggr-scrub-stop
- api-aggr-scrub-suspend
- api-aggr-set-option
- api-aggr-split
- api-aggr-verify-list-info
- api-aggr-verify-resume
- api-aggr-verify-start
- api-aggr-verify-stop
- api-aggr-verify-suspend
- api-cf-*
- api-cf-force-takeover
- api-cf-get-partner
- api-cf-giveback
- api-cf-negotiated-failover-disable
- api-cf-negotiated-failover-enable
- api-cf-negotiated-failover-status
- api-cf-service-disable
- api-cf-service-enable
- api-cf-status
- api-cf-takeover
- api-cifs-*
- api-cifs-homedir-path-get-for-user
- api-cifs-session-list-iter-end

- api-cifs-session-list-iter-next
- api-cifs-session-list-iter-start
- api-cifs-share-add
- api-cifs-share-change
- api-cifs-share-delete
- api-cifs-share-list-iter-end
- api-cifs-share-list-iter-next
- api-cifs-share-list-iter-start
- api-disk-*
- api-disk-fail
- api-disk-list-info
- api-disk-remove
- api-disk-replace-start
- api-disk-replace-stop
- api-disk-sanown-assign
- api-disk-sanown-filer-list-info
- api-disk-sanown-list-info
- api-disk-sanown-reassign
- api-disk-unfail
- ems-autosupport-log
- api-fc-*
- api-fc-config-adapter-disable
- api-fc-config-adapter-enable
- api-fc-config-list-info
- api-fc-config-list-iter-end
- api-fc-config-list-iter-next
- api-fc-config-list-iter-start
- api-fc-config-set-adapter-fc-type
- api-fcp-*
- api-fcp-adapter-clear-partner
- api-fcp-adapter-config-down
- api-fcp-adapter-config-media-type
- api-fcp-adapter-config-up
- api-fcp-adapter-initiators-list-info
- api-fcp-adapter-list-info
- api-fcp-adapter-reset-stats
- api-fcp-adapter-set-partner
- api-fcp-adapter-stats-list-info
- api-fcp-get-cfmode
- api-fcp-node-get-name
- api-fcp-node-set-name
- api-fcp-service-start
- api-fcp-service-status
- api-fcp-service-stop
- api-fcp-set-cfmode
- api-file-*
- api-file-create-symlink
- api-file-delete-directory
- api-file-delete-file
- api-file-get-space-reservation-info
- api-file-list-directory-iter-end
- api-file-list-directory-iter-next
- api-file-list-directory-iter-start
- api-file-read-file
- api-file-read-symlink
- api-file-set-space-reservation-info
- api-file-truncate-file
- api-file-write-file
- api-fpolicy-*

- api-fpolicy-create-policy
- api-fpolicy-destroy-policy
- api-fpolicy-disable
- api-fpolicy-disable-policy
- api-fpolicy-enable
- api-fpolicy-enable-policy
- api-fpolicy-extensions
- api-fpolicy-extensions-list-info
- api-fpolicy-get-required-info
- api-fpolicy-get-secondary-servers-info
- api-fpolicy-list-info
- api-fpolicy-server-list-info
- api-fpolicy-server-stop
- api-fpolicy-set-required
- api-fpolicy-set-secondary-servers
- api-fpolicy-status
- api-igroup-*
- api-igroup-add
- api-igroup-create
- api-igroup-destroy
- api-igroup-list-info
- api-igroup-lookup-lun
- api-igroup-remove
- api-igroup-set-attribute
- api-ipspace-list-info
- api-iscsi-*
- api-iscsi-adapter-config-down
- api-iscsi-adapter-config-up
- api-iscsi-adapter-initiators-list-info
- api-iscsi-adapter-list-info
- api-iscsi-adapter-reset-stats
- api-iscsi-adapter-stats-list-info
- api-iscsi-auth-generate-chap-password
- api-iscsi-initiator-add-auth
- api-iscsi-initiator-auth-list-info
- api-iscsi-initiator-delete-auth
- api-iscsi-initiator-get-auth
- api-iscsi-initiator-get-default-auth
- api-iscsi-initiator-set-default-auth
- api-iscsi-isns-config
- api-iscsi-isns-get-info
- api-iscsi-isns-start
- api-iscsi-isns-stop
- api-iscsi-isns-update
- api-iscsi-node-get-name
- api-iscsi-node-set-name
- api-iscsi-service-start
- api-iscsi-service-status
- api-iscsi-service-stop
- api-license-*
- api-license-add
- api-license-delete
- api-license-list-info
- api-lun-*
- api-lun-clone-split-start
- api-lun-clone-split-status-list-info
- api-lun-clone-split-stop
- api-lun-clone-start
- api-lun-clone-status-list-info

- api-lun-clone-stop
- api-lun-config-check-cfmode-info
- api-lun-config-check-info
- api-lun-create-by-size
- api-lun-create-clone
- api-lun-create-from-file
- api-lun-create-from-snapshot
- api-lun-destroy
- api-lun-get-attribute
- api-lun-get-comment
- api-lun-get-geometry
- api-lun-get-maxsize
- api-lun-get-minsize
- api-lun-get-select-attribute
- api-lun-get-serial-number
- api-lun-get-space-reservation-info
- api-lun-has-scsi-reservations
- api-lun-initiator-logged-in
- api-lun-list-info
- api-lun-map
- api-lun-map-list-info
- api-lun-move
- api-lun-offline
- api-lun-online
- api-lun-port-has-scsi-reservations
- api-lun-reset-stats
- api-lun-resize
- api-lun-restore-status
- api-lun-set-attribute
- api-lun-set-comment
- api-lun-set-select-attribute
- api-lun-set-serial-number
- api-lun-set-share
- api-lun-set-space-reservation-info
- api-lun-snap-usage-list-info
- api-lun-stats-list-info
- api-lun-unmap
- api-lun-unset-attribute
- api-nameservice-*
- api-nameservice-map-gid-to-group-name
- api-nameservice-map-group-name-to-gid
- api-nameservice-map-sid-to-uid
- api-nameservice-map-uid-to-user-name
- api-nameservice-map-unix-to-windows
- api-nameservice-map-user-name-to-uid
- api-nameservice-map-windows-to-unix
- api-nfs-*
- api-nfs-disable
- api-nfs-enable
- api-nfs-exportfs-append-rules
- api-nfs-exportfs-append-rules-2
- api-nfs-exportfs-check-permission
- api-nfs-exportfs-delete-rules
- api-nfs-exportfs-fence-disable
- api-nfs-exportfs-fence-enable
- api-nfs-exportfs-flush-cache
- api-nfs-exportfs-list-rules
- api-nfs-exportfs-list-rules-2
- api-nfs-exportfs-load-exports

- api-nfs-exportfs-modify-rule
- api-nfs-exportfs-modify-rule-2
- api-nfs-exportfs-storage-path
- api-nfs-get-supported-sec-flavors
- api-nfs-monitor-ignore
- api-nfs-monitor-list
- api-nfs-monitor-reclaim
- api-nfs-monitor-watch
- api-nfs-status
- api-options-*
- api-options-get
- api-options-list-info
- api-options-set
- api-perf-*
- api-perf-object-counter-list-info
- api-perf-object-get-instances
- api-perf-object-get-instances-iter-end
- api-perf-object-get-instances-iter-next
- api-perf-object-get-instances-iter-start
- api-perf-object-instance-list-info
- api-perf-object-instance-list-info-iter-end
- api-perf-object-instance-list-info-iter-next
- api-perf-object-instance-list-info-iter-start
- api-perf-object-list-info
- api-qtree-*
- api-qtree-create
- api-qtree-list
- api-qtree-list-iter-end
- api-qtree-list-iter-next
- api-qtree-list-iter-start
- api-quota-*
- api-quota-add-entry
- api-quota-delete-entry
- api-quota-get-entry
- api-quota-list-entries
- api-quota-list-entries-iter-end
- api-quota-list-entries-iter-next
- api-quota-list-entries-iter-start
- api-quota-modify-entry
- api-quota-off
- api-quota-on
- api-quota-report
- api-quota-report-iter-end
- api-quota-report-iter-next
- api-quota-report-iter-start
- api-quota-resize
- api-quota-set-entry
- api-quota-status
- api-reallocate-*
- api-reallocate-delete-schedule
- api-reallocate-list-info
- api-reallocate-off
- api-reallocate-on
- api-reallocate-quiesce
- api-reallocate-restart
- api-reallocate-set-schedule
- api-reallocate-start
- api-reallocate-stop
- api-snapmirror-*

- api-snapmirror-abort
- api-snapmirror-break
- api-snapmirror-delete-connection
- api-snapmirror-delete-schedule
- api-snapmirror-delete-sync-schedule
- api-snapmirror-get-status
- api-snapmirror-get-volume-status
- api-snapmirror-initialize
- api-snapmirror-list-connections
- api-snapmirror-list-destinations
- api-snapmirror-list-schedule
- api-snapmirror-list-sync-schedule
- api-snapmirror-off
- api-snapmirror-on
- api-snapmirror-quiesce
- api-snapmirror-release
- api-snapmirror-resume
- api-snapmirror-resync
- api-snapmirror-set-connection
- api-snapmirror-set-schedule
- api-snapmirror-set-sync-schedule
- api-snapmirror-update
- api-snapshot-*
- api-snapshot-create
- api-snapshot-delete
- api-snapshot-delta-info
- api-snapshot-get-reserve
- api-snapshot-get-schedule
- api-snapshot-list-info
- api-snapshot-reclaimable-info
- api-snapshot-rename
- api-snapshot-reserve-list-info
- api-snapshot-restore-file
- api-snapshot-restore-volume
- api-snapshot-set-reserve
- api-snapshot-set-schedule
- api-snapshot-volume-info
- api-snmp-*
- api-snmp-community-add
- api-snmp-community-delete
- api-snmp-community-delete-all
- api-snmp-disable
- api-snmp-enable
- api-snmp-get
- api-snmp-get-next
- api-snmp-status
- api-snmp-trap-disable
- api-snmp-trap-enable
- api-snmp-traphost-add
- api-snmp-traphost-delete
- api-system-*
- api-system-get-info
- api-system-get-ontapi-version
- api-system-get-version
- api-useradmin-*
- api-useradmin-group-add
- api-useradmin-group-delete
- api-useradmin-group-list
- api-useradmin-group-modify

- api-useradmin-role-add
- api-useradmin-role-delete
- api-useradmin-role-list
- api-useradmin-role-modify
- api-useradmin-user-add
- api-useradmin-user-delete
- api-useradmin-user-list
- api-useradmin-user-modify
- api-useradmin-user-modify-password
- api-vfiler-*
- api-vfiler-add-ipaddress
- api-vfiler-add-storage
- api-vfiler-allow-protocol
- api-vfiler-create
- api-vfiler-destroy
- api-vfiler-disallow-protocol
- api-vfiler-get-allowed-protocols
- api-vfiler-get-disallowed-protocols
- api-vfiler-get-status
- api-vfiler-list-info
- api-vfiler-migrate
- api-vfiler-remove-ipaddress
- api-vfiler-remove-storage
- api-vfiler-setup
- api-vfiler-start
- api-vfiler-stop
- api-volume-*
- api-volume-add
- api-volume-clone-create
- api-volume-clone-split-estimate
- api-volume-clone-split-start
- api-volume-clone-split-status
- api-volume-clone-split-stop
- api-volume-container
- api-volume-create
- api-volume-destroy
- api-volume-get-filer-info
- api-volume-get-language
- api-volume-get-root-name
- api-volume-list-info
- api-volume-mediascrub-list-info
- api-volume-mirror
- api-volume-offline
- api-volume-online
- api-volume-options-list-info
- api-volume-rename
- api-volume-restrict
- api-volume-scrub-list-info
- api-volume-scrub-resume
- api-volume-scrub-start
- api-volume-scrub-stop
- api-volume-scrub-suspend
- api-volume-set-language
- api-volume-set-option
- api-volume-set-total-files
- api-volume-size
- api-volume-split
- api-volume-verify-list-info
- api-volume-verify-resume

- api-volume-verify-start
- api-volume-verify-stop
- api-volume-verify-suspend
- api-volume-wafl-info[BEZM3]