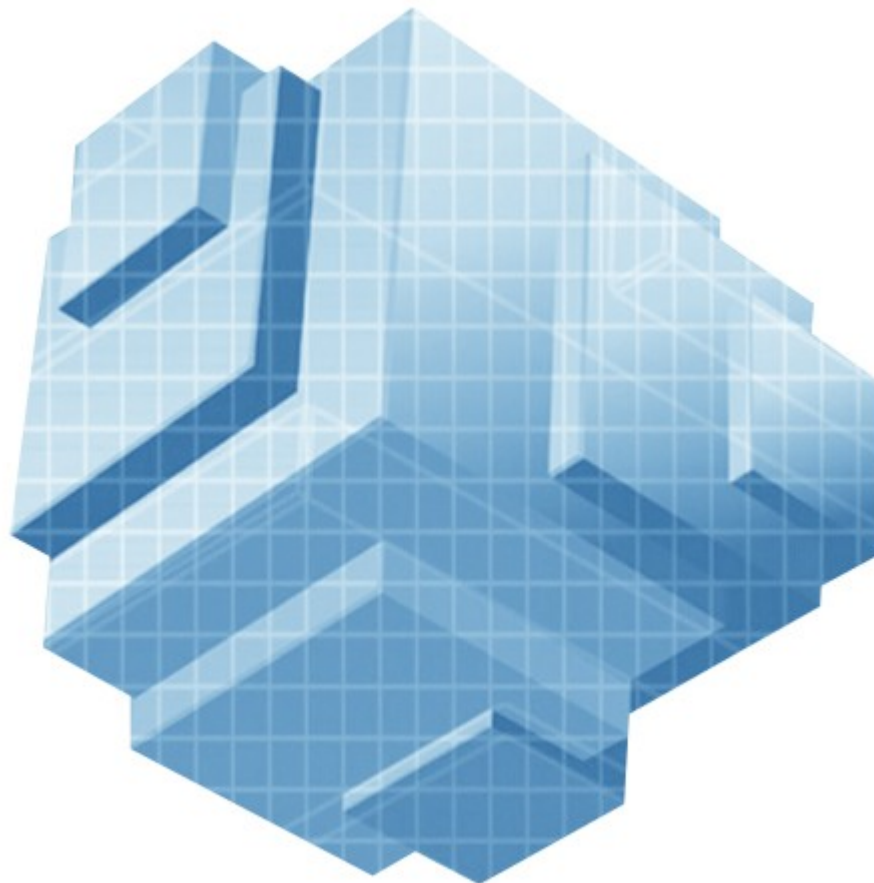




**PresSTORE Command-Language Interface
Reference Manual for
PresSTORE 3.2.0 and above**





Content

The nsdchat utility.....	4
The libchat library.....	6
PresSTORE CLI command summary.....	7
Resource independent commands.....	7
geterror.....	7
srvinfo.....	7
Summary of resource commands accepted by the CLI.....	9
Client.....	9
Workstation.....	10
Server.....	13
ArchiveIndex.....	18
ArchivePlan.....	21
BackupPlan.....	25
SyncPlan.....	27
SyncSelection.....	30
ArchiveSelection.....	33
RestoreSelection.....	37
ArchiveEntry.....	41
Volume.....	45
Jukebox.....	48
Device.....	50
Job.....	51
Pool.....	54
Example CLI usage.....	56

Document revision 03/10



The PresSTORE CLI (Command Language Interface) is a means of accessing the PresSTORE command language, as implemented within the PresSTORE application server.

The CLI allows you to create, interrogate, modify and destroy various PresSTORE resources. A resource is, for example, client, filter, backup plan, archive plan, and alike. Resources are tracked in the PresSTORE configuration database.

The CLI can be accessed in several ways, ranging from simple shell-scripts running on the same computer as the PresSTORE server to networked applications running on any computer, located anywhere on the Internet. There are basically two vehicles offering access to the CLI. This first is the standalone **nsdchat** utility which is included in the standard PresSTORE distribution. The second is the **libchat** library you can use to link with a C-program.



The nsdchat utility

This is the Unix command-line program which you can use to gain access to the CLI from shell scripts. The `nsdchat` utility is located in the `bin/` subdirectory under the PresSTORE installation directory.

The general syntax of the `nsdchat` utility is:

```
nsdchat [options] cli_command [args]
```

The `options` denotes variable number of command options

The `args` denotes variable number of arguments. The entries surrounded in brackets are optional.

The relevant option of the `nsdchat` in respect to the CLI is the `-c` option. The `-c` option executes the CLI command with optional arguments on the default PresSTORE server, like for example:

```
nsdchat -c ArchivePlan names
```

The above CLI command lists names of all known archive plans located on the default PresSTORE server running on the local computer.

By using the `-s` option of the `nsdchat` utility you can specify some other PresSTORE server than the default. There are two ways to specify the server, depending on the communication method used. The "nsdchat" supports two communication methods, named pipes or TCLsockets

Named-pipes can be used only when the `nsdchat` utility and the PresSTORE server are running on the same computer. TCP sockets can be used for both local and network-wide connections.

Depending on the selected communication mode, the `-s` option of the "nsdchat utility, the connection identifier, might take following forms:

For TCP sockets:

```
awsock: /<user>:<passwd>:<session>@<host>:<port>
```

<user>	required	name of the user
<password>	required	users password
<session>	optional	session identifier (see hints below)
<host>	required	hostname or IP address of the PresSTORE host
<port>	required	port number of the PresSTORE socket server

Examples:

```
nsdchat -s awsock:/user:passw@my.host.com:9001 -c srvinfo version
nsdchat -s awsock:/user:pass:311@my.host.com:9001 -c srvinfo version
```



For named-pipes:

```
awfile: /<homedir>:<server>
```

or

```
awfile: /<user>:<password>:<session>@<homedir>:<server>
```

<user>	optional	name of the user
<password>	optional	users password
<session>	optional	session identifier (see hints below)
<homedir>	required	PresSTORE installation directory
<server>	required	currently, only <code>lexxsrv</code> is allowed

Note: the user who executes the `nsdchat` utility is taken as identification when logging onto the PresSTORE server, unless you provide the username and password as part of the connection identifier.

Examples

```
nsdchat -s awfile://usr/local/aw:lexxsrv -c srvinfo version
nsdchat -s awfile:/s-10@usr/local/aw:lexxsrv -c srvinfo version
```

In the above examples, one of the connection string elements, the `session` deserves some extra clarification.:

Normally, for each CLI connection, there is a server-side session maintained. If you start two or more `nsdchat` sessions for the same username (by running two `nsdchat` programs or two programs linked with `libnsdchat` library) then both will be using the same session on the server, effectively trampling on each other's toes, i.e. you will have a session clash. In order to avoid this, give each instance of `nsdchat` call an unique ID. This unique ID will be used to create and identify the correct server-side session.

The `nsdchat` utility has an option to read and execute CLI commands from within a file. For this kind of operation, specify the name of the file on the `nsdchat` command line:

```
nsdchat mycommands.cli
```

and all commands in the `mycommands.cli` will be executed as a unit on the default PresSTORE server. In addition to that, you can also make `mycommands.cli` file an executable program on unix systems by setting the appropriate privilege mask and making the first line of the file look like:

```
#!/usr/local/aw/bin/nsdchat
```

Please note that in this example, the PresSTORE installation directory is given as default: `/usr/local/aw`. This may vary in your particular case. If it does, replace the `/usr/local/aw` with the correct location of the installation directory.



The libchat library

You normally link this library with your C-program to gain access to the CLI. The library exposes a very simple API which you can use to log-in to the PresSTORE server, send commands and process results. The library is available for all supported PresSTORE platforms in both static and shared object form.

The library offers the same functionality as the "nsdchat" utility with one notable exception: the library will allow you to handle events. PresSTORE usually sends events to all logged users each time some resource gets changed, created or deleted. By using API calls from the C-library you can register event handlers which will be invoked for each event received on the communication link.

Environmental variables

The following environmental variables are generated by PresSTORE:

Variables set on the PresSTORE server:

AWPST_CLN_HOST	name of the PresSTORE client host
AWPST_CLN_PORT	port of the PresSTORE client
AWPST_CLN_PCLI	port for the client CLI communication
AWPST_CLN_HOME	installation home directory

Variables set on the PresSTORE client:

AWPST_SRV_HOST	name of the PresSTORE server host
AWPST_SRV_PORT	port of the PresSTORE server
AWPST_SRV_PCLI	port for the server CLI communication
AWPST_SRV_HOME	installation home directory
AWPST_SRV_JOB	name of the job running on the server

These environmental variables are defined in pre- and post-scripts invoked by PresSTORE when processing various jobs.



PresSTORE CLI command summary

The CLI consists of a set of commands one can use to manipulate resources and initiate and control various data-management tasks. All commands of the CLI have the same base syntax:

```
cli_command method resource [parameter [value]...]
```

or

```
cli_command resource-name method [parameter [value]...]
```

The "cli_command" is the name of the resource command. The resource command accepts one mandatory argument, which is either the existing resource name, or one of general or resource specific subcommands, as described below. Both forms accept variable number of `arg/value` pairs.

All CLI commands return empty result ("") in case of error. To find-out the real cause of the error (i.e. display the error message) you can use the `geterror` CLI command. In addition to resource commands, there are other, resource-independent commands which operate on the global level.

The CLI is built atop the well-known extension language Tcl. It understands all Tcl control structures, so you can write full-fledged programs. The CLI interpreter runs in Tcl safe-interpreter mode. See <http://www.tcl.tk> for more information about the Tcl language.

Resource independent commands

geterror

Returns the error message associated with the last issued CLI command. You should invoke this command after getting the empty result string from any CLI command(s) in order to get the explanation of the error encountered.

srvinfo

This command returns information about the current PresSTORE server.

Method:	hostname
Syntax:	srvinfo hostname
Description:	Returns hostname of the PresSTORE host
Return values:	The hostname as returned with <code>hostname</code> shell command

Method:	address
Syntax:	srvinfo address
Description:	Returns IP address of the PresSTORE host
Return values:	The IP address in dotted notation



Method: **port**
Syntax: `svinfo port`
Description: Returns TCP port of the PresSTORE server
Return values: The TCP port number

Method: **server**
Syntax: `svinfo server`
Description: Returns the name of the PresSTORE server. Currently there is only one server assigned: lexxsrv.
Return values: The server name

Method: **platform**
Syntax: `svinfo platform`
Description: Returns the OS platform of the PresSTORE host
Return values: One of: linux, solaris, windows or macosx

Method: **version**
Syntax: `svinfo version`
Description: Returns the version of the PresSTORE application server.
Return values: The application server version string as X.Y number

Method: **lexxvers**
Syntax: `svinfo lexxvers`
Description: Returns the PresSTORE application version
Return values: The application version string as X.Y.Z number



Summary of resource commands accepted by the CLI

Client

Inquiries configured PresSTORE clients and their parameters. PresSTORE client is the computer running PresSTORE client software. PresSTORE server is the computer running PresSTORE server software. One server can archive, backup, restore and synchronize files to and from any registered client(s).

In the current version of the CLI, you only have read access to client data. You can't modify any of the existing clients nor can you create new or delete existing clients. Client resources are configured and maintained with standard PresSTORE Web GUI by the system administrator.

Method:	names
Syntax:	Client names
Description:	Returns list of names of all clients.
Return values:	On success: the list of names On failure: an empty string

Method:	describe
Syntax:	Client <name> describe
Description:	Returns human-readable description of the client <name>. If the client has no description assigned, the command returns the string: "<empty>"
Return values:	On success: the client description On failure: an empty string

Method:	hostname
Syntax:	Client <name> hostname
Description:	Returns hostname (or IP address) of the client <name>
Return values:	On success: the hostname or IP address On failure: an empty string

Method:	port
Syntax:	Client <name> port
Description:	Returns the TCP port of the client <name>
Return values:	On success: the configured TCP port On failure: an empty string



Method:	ping
Syntax:	Client <name> ping [<timeout>]
Description:	Tests the connection to the <name> client. The optional <timeout> argument controls how many seconds to wait for the client response. If the argument is omitted, the timeout defaults to 600 seconds (10 minutes).
Return values:	-4 wrong client version -3 client is disabled -2 wrong username/password -1 network connection problem 0 (reserved for future use) 1 ping OK

Workstation

Inquiries configured PresSTORE Backup2Go workstations and their parameters. PresSTORE workstation is the computer running PresSTORE client software. In current version of the CLI, you only have read access to workstations data. You can't modify any of the existing workstations nor can you create new or delete existing workstations. Workstations resources are configured and maintained with standard PresSTORE Web GUI by the system administrator.

Method:	names
Syntax:	Workstation names
Description:	Returns list of names of all workstations
Return values:	On success: the list of names On failure: an empty string

Method:	enable
Syntax:	Workstation <name> enable
Description:	Sets the workstation to the <code>Enabled</code> state
Return values:	On success: the string "1" On failure: an empty string



Method: **disable**
Syntax: Workstation <name> disable
Description: Sets the workstation to the `Disabled` state
Return values: On success: the string "0"
On failure: an empty string

Method: **enabled**
Syntax: Workstation <name> enabled
Description: Queries the workstation `Enabled` status
Return values: 1 enabled
0 disabled
On failure: an empty string

Method: **disabled**
Syntax: Workstation <name> disabled
Description: Queries the workstations `Disabled` status
Return values: 1 disabled
0 enabled
On failure: an empty string

Method: **describe**
Syntax: Workstation <name> describe
Description: Returns human-readable description of the workstation <name>. If the workstation has no description assigned, the command returns the string: "<empty>"
Return values: On success: the workstation description
On failure: an empty string

Method: **peerip**
Syntax: Workstation <name> peerip
Description: Returns last known IP of the workstation <name>. If the workstation has no recorded IP so far (for example, it never got connected to the server), the command returns the string: "<empty>"
Return values: On success: the workstation IP address in standard dot notation
On failure: an empty string



Method: **hostid**
Syntax: Workstation <name> hostid
Description: Returns the configured PresSTOORE machine-ID of the workstation <name>.
Return values: On success: the workstation I machine ID
On failure: an empty string

Method: **totalfiles**
Syntax: Workstation <name> totalfiles
Description: Returns number of files transfered from the workstation <name> in the last backup operation
Return values: On success: the number of files
On failure: an empty string

Method: **totalkbytes**
Syntax: Workstation <name> totalkbytes
Description: Returns number of KBytes transfered from the workstation <name> in the last backup operation
Return values: On success: the number of KBytes
On failure: an empty string

Method: **lastbegin**
Syntax: Workstation <name> lastbegin
Description: Returns absolute time in time_t format (number of seconds since the Epoch) of the start of the last backup operation for the workstations <name>
Return values: On success: the time in time_t format
On failure: an empty string

Method: **lastend**
Syntax: Workstation <name> lastend
Description: Returns absolute time in time_t format (number of seconds since the Epoch) of the sucessfull end of the last backup operation for the workstations <name>. This time may be older then the time returned by "lastbegin" method. This indicates an incomplete (interrupted) backup.
Return values: On success: the time in time_t format
On failure: an empty string



Method: **nextrun**
Syntax: Workstation <name> nextrun
Description: Returns absolute time in time_t format (number of seconds since the Epoch) of the next anticipated backup of the workstation
Return values: On success: the time in time_t format
On failure: an empty string

Method: **configure**
Syntax: Workstation configure <hostname> <port> <username> <password> [<template>]
Description: Run this command on the PresSTORE Backup2Go Server. Using the passed connection parameters <hostname> and <port> , try to establish the connection to the remote workstation and, based on it's host ID, create or reuse the workstation record on the server. For the purpose of logging into the server, the workstation will be seeded with an unique token, shared by the workstation and the server. This eliminates need for storing the <username> and/or <password> for accessing the server on the workstation. If the optional <template> is given the workstation is set to use the given template. Otherwise the workstation is set to use the generic template. This method is introduced in PresSTORE 3.2
Return values: On success: a positive integer (name of the new local workstation)
On failure: "-3": template could not be set
"-2": wrong username/password
"-1": network connection problem (bad address and/or port)

Server

Inquiries configured PresSTORE Backup2Go server's and their parameters. PresSTORE server is the computer running PresSTORE server software and providing backup services for PresSTORE workstation computers.

Method: **names**
Syntax: Server names
Description: Returns list of names of all configured servers
Return values: On success: list of names
On failure: an empty string



Method: **create**
Syntax: Server create
Description: Creates new server resource
Return values: On success: name/ID of the new server resource
On failure: an empty string

Method: **delete**
Syntax: Server <name> delete
Description: Deletes server resource, automatically stopping any scheduled job. If any jobs are running, the resource will not be deleted
Return values: On success: 1 if deleted or 0 if not
On failure: an empty string

Method: **enable**
Syntax: Server <name> enable
Description: Sets the server to the "Enabled" state automatically starting scheduled job
Return values: On success: string "1"
On failure: an empty string

Method: **disable**
Syntax: Server <name> disable
Description: Sets the server to the "Disabled" state automatically stopping any scheduled job
Return values: On success: string "0"
On failure: an empty string

Method: **enabled**
Syntax: Server <name> enabled
Description: Queries the server `Enabled` status.
Return values: On success: string "1" (enabled) or "0" (not enabled)
On failure: an empty string

Method: **disabled**
Syntax: Server <name> disabled
Description: Queries the server `Disabled` status
Return values: On success: string "1" (disabled) or "0" (not disabled)
On failure: an empty string



On failure: an empty string

Method: ping

Syntax: Server <name> ping [<timeout>]

Description: Tests the connection to the <name> server. The optional <timeout> argument controls how many seconds to wait for the servers response. If the argument is omitted, the timeout defaults to 600 seconds (10 minutes).

Return values:

- 2 wrong username/password
- 1 network connection problem
- 0 reserved for future use)
- 1 ping ok

Method: username

Syntax: Server <name> username [<value>]

Description: If no additional arguments specified, returns the name of the user used for authentication on the current server. Otherwise interprets the given argument as the new user name and stores the value.

Return values:

- On success: the user name
- On failure: an empty string

Method: password

Syntax: Server <name> password [<value>]

Description: If no additional arguments specified, returns the password of the user used for authentication on the current server. Otherwise interprets the given argument as the new password and stores the value.

Return values:

- On success: the password
- On failure: an empty string

Method: hostname

Syntax: Server <name> hostname [<value>]

Description: If no additional arguments specified, returns the hostname or IP address of the server. Otherwise interprets the given argument as the new hostname and stores the value

Return values:

- On success: the hostname
- On failure: an empty string

Method: port



Description: If no additional arguments specified, returns the TCP port number of the server. Otherwise interprets the given argument as the new port number and stores the value

Return values: On success: the port number
On failure: an empty string

Method: **pathlist**

Syntax: Server<name> pathlist [<value>]

Description: If no additional arguments specified, returns the list of paths configured for the backup operation. Paths are delimited by a single space character. If one of the returned paths itself contains one or more spaces, the complete path is enclosed in curly braces { and } . Otherwise interprets the given argument as the new list of paths and stores the value. Each path in the list must be delimited from the follower by a single space. If one of the given paths itself contains one or more spaces, that whole path must be enclosed in curly braces { and } .

Return values: On success: list of paths separated by a single space
On failure: an empty string

Method: **throttle**

Syntax: Server <name> throttle [<value>]

Description: If no additional arguments specified, returns the bandwidth throttle in percent (0% - 100%) of the communication link used to talk to this server. Otherwise interprets the given argument as the new throttle value and stores the value.

Return values: On success: the throttle value in percent
On failure: an empty string

Method: **useevents**

Syntax: Server <name> useevents [<value>]

Description: If no additional arguments specified, returns the boolean flag (1 or 0) depending whether the workstation will use FSEvents facility when gathering files to be stored on this server (1) or will revert to the linear filesystem walk (0). Otherwise interprets the given argument as the new flag value and stores the value

Return values: On success: the boolean flag (0 or 1)
On failure: an empty string



Method: **usecompression**

Syntax: Server <name> usecompression [<value>]

Description: If no additional arguments specified, returns the boolean flag (1 or 0) depending whether the workstation will compress the network traffic targeted to this server (1) or not (0). Otherwise interprets the given argument as the new flag value and stores the value.

Return values: On success: the boolean flag (0 or 1)
On failure: an empty string

Method: **netencryption**

Syntax: Server <name> netencryption [<value>]

Description: If no additional arguments specified, returns the boolean flag (1 or 0) depending whether the workstation will encrypt the network traffic targeted to this server (1) or not (0). Otherwise interprets the given argument as the new flag value and stores the value.

Return values: On success: the boolean flag (0 or 1)
On failure: an empty string

Method: **dataencryption**

Syntax: Server <name> dataencryption [<value>]

Description: If no additional arguments specified, returns the boolean flag (1 or 0) depending whether the workstation will encrypt file contents of the files transferred to this server and store them on the server in encrypted form (1) or not (0). Otherwise interprets the given argument as the new flag value and stores the value.

Return values: On success: the boolean flag (0 or 1)
On failure: an empty string

Method: **reschedule**

Syntax: Server <name> reschedule [<value>]

Description: If no additional arguments specified, returns the number of hours to re-schedule the backup job after the regular completion. Note that non-regularly completed jobs are automatically rescheduled immediately. Otherwise interprets the given argument as the new number of hours and stores the value.

Return values: On success: the number of hours
On failure: an empty string



ArchiveIndex

Inquiries PresSTORE archive index databases and their parameters. Archive index databases are used to track information about archived files, their location on the storage media, user-defined metadata and related information.

In the current version of the CLI, you only have limited write access to archive index databases. You can modify some configuration details of the existing and you can create new archive index databases. If you need a full control of ArchiveIndex resources, please use the PresSTORE Web GUI.

Method:	names
Syntax:	ArchiveIndex names
Description:	Returns list of names of archive indexes.
Return values:	On success: list of names. If no archive indexes are configured, command returns the string: "<empty>" On failure: an empty string

Method:	create
Syntax:	ArchiveIndex create <name> <description>
Description:	Creates <name> archive index database with <description>. If an archive index with the same <name> already exists, an error is thrown. The <name> should not contain blanks, special intepunction characters nor any special national characters. The <description> may contain any text.
Return values:	On success: name of the newly created index database On failure: an empty string

Method:	keys
Syntax:	ArchiveIndex <name> keys
Description:	Reports all user-defined meta keys for the index <name>. Meta keys are used to store user-given metadata to selected elements in the archive index.
Return values:	On success: list of keys. If no keys have been defined the command returns the string: "<empty>" On failure: an empty string



Method: **addkey**

Syntax: ArchiveIndex <name> addkey <key> <type> [<attr value> ...]

Description: Adds one user-defined key in the given index. The <key> identifier should not contain blanks, special interpunction characters nor any national characters. The length of the <key> identifier should not exceed 15 characters. The <type> designates the data type reserved for the <key>. It must be one of:

- C character key
- N numeric key

This command also accepts a variable number of user given attributes and their values attached to the <key>. Both <attr> and <value> may contain any characters but the length of each of them is limited to 15 characters. These entries are optional and are not interpreted by PresSTORE in any way, except stored in the key definition in the archive index.

Return values: On success: names of all configured keys
On failure: an empty string

Method: **delkey**

Syntax: ArchiveIndex <name> delkey <key>

Description: Deletes one user-defined key in the given index

Return values: On success: names of all configured keys
On failure: an empty string

Method: **keyhas**

Syntax: ArchiveIndex <name> keyhas <key> <attr>

Description: Checks wether the <key> has attribute <attr> defined

Return values: On success: True (1) if yes, false (0) otherwise
On failure: an empty string



Method: **keyget**

Syntax: ArchiveIndex <name> keyget <key> [<attr>]

Description: Returns attributes for the given <key>. If no optional <attr> is supplied, returns all defined attributes and their values as list of key/value pairs. If the <attr> is supplied, returns the value of the <attr> attribute.
Each <key> has at least one attribute: "type". Please see [the addkey method](#) for description of the "type" attribute.

Return values: On success: either a list of all defined attributes and values for the given <key>, or just the attribute value, depending on the existence of the optional argument <attr>
On failure: an empty string

Method: **keyset**

Syntax: ArchiveIndex <name> keyset <key> <attr> <val>

Description: Sets the value <val> of the user-given attribute <attr> for the given <key>. The value of the attribute `type` cannot be set. Please see the [addkey method](#) for description of the `type` attribute.

Return values: On success: 1 (true) if the <attr> has been set to the given value <val> or 0 (false) if the key could not be set (the <key> was not found, for example)
On failure: an empty string

Method: **backup**

Syntax: ArchiveIndex <name> backup <filename>

Description: Produces the backup of the <name> archive index and leaves the backup file at the <filename>.

Return values: On success: the filename of the backup file
On failure: an empty string

Method: **restore**

Syntax: ArchiveIndex <name> restore <filename>

Description: Restores the archive database <name> from the given <filename>. The <filename> must be the one used to produce backup of the database (see method `backup`).

Return values: On success: the filename of the backup file
On failure: an empty string

ArchivePlan

Manages PresSTORE archive plan(s) and their parameters. Archive plans are used to group various parameters of the archive operation, like selected index database, pool of media, time schedule and various other details. PresSTORE administrator defines archive plans according to the custom site policies. User who wishes to archive files must select one of the predefined archive plans.

In current version of the CLI, you only have limited write access to archive plans. You can modify some configuration details of existing and you can create new archive plans. If you need a full control of ArchivePlan resources configuration, please use the PresSTORE Web GUI.

Method: **names**
Syntax: ArchivePlan names
Description: Returns list of names of all configured ArchivePlan
Return values: On success: list of plan names. If no plans have been configured, the command returns the string: "<empty>
On failure: an empty string

Method: **enable**
Syntax: ArchivePlan <name> enable
Description: Sets the plan to the "Enabled" state
Return values: On success: string "1"
On failure: an empty string

Method: **disable**
Syntax: ArchivePlan <name> disable
Description: Sets the plan to the Disabled state
Return values: On success: string "0"
On failure: an empty string

Method: **enabled**
Syntax: ArchivePlan <name> enabled
Description: Queries the plan Enabled status
Return values: On success: "1" (plan is enabled) or "0" (not enabled)
On failure: an empty string



Method: **disabled**
Syntax: ArchivePlan <name> disabled
Description: Queries the plan Disabled status
Return values: On success: "1" (plan is disabled) or "0" (not disabled)
On failure: an empty string

Method: **create**
Syntax: ArchivePlan <description>
Description: Creates a new archive plan with the given <description>. If an archive plan with the same <description> already exists, an error is thrown.

The newly created plan might be further configured for operation by using the `database`, `pool` and/or `copypool` methods described below.

If not further configured, the newly generated plan will per-default use the Default-Archive pool and the Default-Archive database.
Return values: On success: name of the newly created plan
On failure: an empty string

Method: **describe**
Syntax: ArchivePlan <name> describe
Description: Returns human-readable description of the archive plan <name>.
Return values: On success: the plan description. If no description has been set the command returns the string: "<empty>"
On failure: an empty string



Method:	database
Syntax:	ArchivePlan <name> database [<value>]
Description:	<p>Returns or sets the name of the index database resource associated with the archive plan <name></p> <p>If the optional <value> argument is not given, the name of the currently configured database will be returned.</p> <p>If the optional <value> argument is given it will be interpreted as the name of the existing archive index database, and the plan <name> will be configured to use the given database. If the referenced database is not configured or disabled an error will be thrown.</p> <p>Also, if the given database is not an archive index, an error will be thrown. You can use the <code>ArchiveIndex</code> resource commands to inspect and/or create archive index databases.</p> <p>Note that ArchivePlan requires that a database is set. Otherwise, the archive job for this plan will fail.</p>
Return values:	<p>On success: the name of the archive index database. If none has been set, the command returns: "<empty>".</p> <p>On failure: an empty string</p>

Method:	pool
Syntax:	ArchivePlan <name> pool [<value>]
Description:	<p>Returns name of the primary media pool resource associated with the archive plan <name>. If the optional <value> argument is not given, the name of the currently configured pool will be returned.</p> <p>If the optional <value> argument is given it will be interpreted as the name of the existing media pool and the plan <name> will be configured to use the given pool instead. If the referenced media pool is not configured, an error will be thrown. Also, if the referenced media pool is not setup for archive operation, or the referenced pool is the same as already configured <code>copypool</code>, an error will be thrown. You can use the "Pool" resource commands to inspect and/or create media pools.</p> <p>Note that ArchivePlan needs the primary media pool set. If none set, the archive job configured to use this plan will fail.</p>
Return values:	<p>On success: the name of the primary media pool. If not configured, it returns the string: "<empty>".</p> <p>On failure: an empty string</p>



Method: **copypool (DEPRECATED)**

Syntax: ArchivePlan <name> copypool [<value>]

Description: Returns name of the secondary media pool resource associated with the archive plan <name>. This pool of media will hold the copy of data from the primary media pool.
If the optional <value> argument is not given, the name of the currently configured copypool will be returned.
If the optional <value> argument is given it will be interpreted as the name of the existing media pool and the plan <name> will be configured to use the given pool instead. If the referenced media pool is not configured, an error will be thrown. Also, if the referenced media pool is not setup for archive operation, or the referenced pool is the same as already configured master pool an error will be thrown. You can use the "Pool" resource commands to inspect and/or create media pools.

Return values: On success: the name of the secondary media pool. If not configured, the string <empty>
On failure: an empty string

Method: **submit**

Syntax: ArchivePlan <name> submit [<now>]

Description: Submits the archive plan for execution. You can optionally override plan execution times by giving the <now> argument as the verbatim string "now" or as the integer value zero.

Note: Please use the returned job ID to query the status of the job by using the `Job` resource. Please see the [Job resource](#) description for more details.

Return values: On success: the archive job ID
On failure: an empty string

Method: **run**

Syntax: ArchivePlan <name> run ?-delete 1?

Description: uns the archive plan immediately with optional delete pass on the target directory/ies.

Note: Please use the returned job ID to query the status of the job by using the `Job` resource. Please see the [Job resource](#) description for more details.

Return values: On success: the archive job ID.
On failure: an empty string



Method: **stop**
Syntax: ArchivePlan <name> stop
Description: Removes the plan <name> from the scheduler
Return values: On success: "1" - plan was successfully removed
"0" - plan was not removed (plan is running)
On failure: an empty string

Method: **cancel**
Syntax: ArchivePlan <name> cancel
Description: Cancels the plan <name> execution. Only running plans can be cancelled. Plans scheduled but not running can be only stopped (see [stop method](#))
Return values: On success: "1" - plan was successfully cancelled
"0" - plan was not cancelled (plan not running)
On failure: an empty string

Method: **verify**
Syntax: ArchivePlan <name> verify <client> <job>
Description: Re-runs the verify, clip generation and deletion (the post-archive tasks) of files located on the <client> computer and archived with the <job> ID.
This method is introduced in PresSTORE 3.2.
Return values: On success: the verify job ID. Use this job ID to query the status of the job by using Job resource. Please see the Job resource description for more details.
"0" - plan was not cancelled (plan not running)
On failure: an empty string

BackupPlan

Inquiries PresSTORE backup plan(s) and their associated parameters. Backup plans are used to group various parameters of the backup operation, like pool of media, time schedules and other details. PresSTORE administrator defines backup plans according to the custom site policies.

In current version of the CLI, you only have read access to backup plans. You can't modify any of the existing plans nor can you create new or delete existing plans. BackupPlan resources are configured and maintained using the PresSTORE Web GUI by the system administrator.



Method: **names**
Syntax: BackupPlan names
Description: Returns list of names of all BackupPlan resources
Return values: On success: list of names. If no backup plans have been configured, the command returns the string: "<empty>".
On failure: an empty string

Method: **enable**
Syntax: BackupPlan <name> enable
Description: Sets the plan to the `Enabled` state
Return values: On success: string "1"
On failure: an empty string

Method: **disable**
Syntax: BackupPlan <name> disable
Description: Sets the plan to the `Disabled` state
Return values: On success: string "0"
On failure: an empty string

Method: **enabled**
Syntax: BackupPlan <name> enabled
Description: Queries the plan `Enabled` status
Return values: On success: "1" (plan is enabled) or "0" (not enabled)
On failure: an empty string

Method: **disabled**
Syntax: BackupPlan <name> disabled
Description: Queries the plan `Disabled` status
Return values: On success: "1" (plan is disabled) or "0" (not disabled)
On failure: an empty string

Method: **describe**
Syntax: BackupPlan <name> describe
Description: Returns human-readable description of the <name> plan. The <name> is one of the elements returned by the "names" method. If the element has no description assigned, the command returns string "<empty>"



Return values: On success: the resource description. If no description has been set the command returns the string: "<empty>"
On failure: an empty string

Method: **submit, start**
Syntax: BackupPlan <name> submit [<now>]
BackupPlan <name> start [<now>]
Description: Submits the backup plan for execution. You can optionally override plan execution times by giving the <now> argument as the verbatim string "now" or as the integer value zero.

Note: Please use the returned job ID to query the status of the job by using the [Job](#) resource. Please see the [Job resource](#) description for more details.
Return values: On success: the backup job ID
On failure: an empty string

Method: **stop**
Syntax: BackupPlan <name> stop
Description: Removes the plan <name> from the scheduler
Return values: On success: "1" - plan was successfully removed
"0" - plan was not removed (plan is running)
On failure: an empty string

Method: **cancel**
Syntax: BackupPlan <name> cancel
Description: Cancels the plan <name> execution. Only running plans can be cancelled. Plans scheduled but not running can be only stopped (see [stop method](#))
Return values: On success: "1" - plan was successfully cancelled
"0" - plan was not cancelled (plan not running)
On failure: an empty string

SyncPlan

Inquiries PresSTORE synchronize plan(s) and their parameters. Sync plans are used to group various parameters of the synchronize operation, like time schedules and various other details. PresSTORE administrator defines sync plans according to the custom site policies.



In the current version of the CLI, you only have read access to sync plans. You can't modify any of the existing plans nor can you create new or delete existing plans. SyncPlan resources are configured and maintained with PresSTORE Web GUI by the system administrator.

Method: **names**
Syntax: SyncPlan names
Description: Returns a list of names of all sync plans
Return values: On success: list of names. If no sync plans have been configured the command returns the string: "<empty>"
On failure: an empty string

Method: **enable**
Syntax: SyncPlan <name> enable
Description: Sets the plan to the `Enabled` state
Return values: On success: string "1"
On failure: an empty string

Method: **disable**
Syntax: SyncPlan <name> disable
Description: Sets the plan to the `Disabled` state
Return values: On success: string "0"
On failure: an empty string

Method: **enabled**
Syntax: SyncPlan <name> enabled
Description: Queries the plan `Enabled` status
Return values: On success: "1" (plan is enabled) or "0" (not enabled).
On failure: an empty string

Method: **disabled**
Syntax: SyncPlan <name> disabled
Description: Queries the plan `Disabled` status
Return values: On success: "1" (plan is disabled) or "0" (not disabled)
On failure: an empty string



Method: **describe**

Syntax: SyncPlan <name> describe

Description: Returns human-readable description of the <name> plan
The <name> is one of the elements returned by the `names` method.
If the element has no description assigned, the command returns string "<empty>".

Return values: On success: the resource description. If no description has been set, the command returns the string: "<empty>"
On failure: an empty string

Method: **sourcehost**

Syntax: SyncPlan <name> sourcehost

Description: Returns the names of the client where the source data is located.

Return values: On success: name of the client
On failure: an empty string

Method: **sourcepath**

Syntax: SyncPlan <name> sourcepath [newpath]

Description: If no optional argument `newpath` specified returns the path of the source directory on the client where the data is located. Otherwise sets the given new path.

Return values: On success: path to the directory
On failure: an empty string

Method: **targethost**

Syntax: SyncPlan <name> targethost

Description: Returns the name of the client where the data should be synced to

Return values: On success: name of the client
On failure: an empty string

Method: **targetpath**

Syntax: SyncPlan <name> targetpath [newpath]

Description: if no optional argument [newpath] specified returns the path of the target directory on the client where the data is to be synced. Otherwise sets the given new path.

Return values: On success: path to the directory
On failure: an empty string



Method: **submit**

Syntax: SyncPlan <name> submit [now]
yncPlan <name> start [now]

Description: Submits the sync plan for execution. You can optionally override plan execution times by giving the <now> argument as the verbatim string "now" or as the integer value zero

Return values: On success: returns the sync job ID. Use this job ID to query the status of the job by using Job resource.
Please see the [Job Resource](#) description for details.
On failure: an empty string

Method: **run**

Syntax: SyncPlan <name> run [delete 1]

Description: Runs the sync plan immediately with optional delete pass on the target directory.

Return values: On success: returns the sync job ID. Use this job ID to query the status of the job by using Job resource.
Please see the [Job resource](#) description for details.
On failure: an empty string

Method: **stop**

Syntax: SyncPlan <name> stop

Description: Removes the plan <name> from the scheduler

Return values: On success: "1" - plan was successfully removed
"0" - plan was not removed (plan is running)
On failure: an empty string

Method: **cancel**

Syntax: SyncPlan <name> cancel

Description: Cancels the plan <name> execution. Only running plans can be canceled. Plans scheduled but not running can be only stopped (see [stop method](#))

Return values: On success: "1" - plan was successfully cancelled
"0" - plan was not cancelled (plan not running)
On failure: an empty string

SyncSelection



The sync selection is used to prepare one or more directories for the sync operation. You can use the resource methods to populate the selection (i.e. add directories) and then submit the entire selection for immediate or scheduled execution.

The sync selection is a temporary resource. It does not survive system crashes and server shutdowns, nor it needs to be explicitly destroyed by the caller. It goes out of scope by invoking the "submit" method, which effectively passes the control to the Job manager. The owner of the sync selection resource is thus the PresSTORE system, so the caller needs not (nor it should) perform any other task with the same resource.

Usage:

To use the `SyncSelection` resource, you must first use the `create` method to create new instance. After the creation, you use the `adddirectory` method to fill-in the selection with directories to synchronize. Finally, you must submit the selection for immediate or scheduled execution. After the submission, the resource goes out of scope and should not be used any more.



Method: **create**

Syntax: SyncSelection create <plan>

Description: Creates new temporary sync selection resource. The resource will be automatically deleted after the associated sync job has been submitted.
The <plan> must be one of the registered synchronize plans. You can get the list of synchronize plans with the "SyncPlan names" CLI command

Return values: On success: The name of the new resource. You must use this name to address the resource in all other methods
On failure: an empty string

Method: **adddirectory**

Syntax: SyncSelection <name> adddirectory <path>

Description: Adds one new directory <path> to the sync selection <name>. It expects the absolute path to the directory to be synced. The directory must be located on the source client and under the source path as given in the sync plan used to create the sync selection object.

Return values: On success: The directory path repeated
On failure: an empty string

Method: **submit**

Syntax: SyncSelection <name> submit [<when>]

Description: Submits the sync selection for execution. You can optionally override plan execution times by giving the <now> as true (one of "1", "t", "true", "True", "y", "yes", or "Yes")
This command implicitly destroys the SyncSelection object for the user and transfers the ownership of the internal underlying object to the job scheduler. You should not attempt to use the <name> afterwards.

Return values: On success: returns the sync job ID. use this job ID to query the status of the job by using Job resource.
Please see the [Job resource](#) description for details.
On failure: an empty string



Method:	destroy
Syntax:	SyncSelection <name> destroy
Description:	Explicitly destroys the sync selection. The <name> should not be used in any SyncSelection commands afterwards.
Return values:	On success: 0 - destroyed, 1 - not destroyed On failure: an empty string

ArchiveSelection

The archive selection is used to prepare one or more files and/or directories for the archive operation. You must create new archive selection resource for each archive session. You can use the resource methods to populate the selection (i.e. add files) and then submit the entire selection for immediate or scheduled execution. The archive selection is a temporary resource. It does not survive system crashes and server shutdowns, nor it needs to be explicitly destroyed by the caller. It goes out of scope by invoking the "submit" method, which effectively passes the control to the Job manager. The owner of the archive selection resource is thus the PresSTORE system, so the caller needs not (nor it should) perform any other task with the same resource.

Usage:

To use the ArchiveSelection resource, you must first use the `create` method to create a new instance. After the creation, use the `addentry` and/or `adddirectory` methods to fill-in the selection with files and/or directories to archive. Finally, submit the selection for immediate or scheduled execution. After the submission, the resource goes out of scope and should not be used any more.



Method:	create
Syntax:	ArchiveSelection create <client> <plan> [<indexroot>]
Description:	<p>Creates new temporary archive selection resource. The resource will be automatically deleted after the associated archive job has been submitted.</p> <p>The <client> must be the one of the registered client computers on the current PresSTORE server. You can get the list of client computers with the <code>Client names</code> CLI command. All files added with the <code>addentry</code> method (below) must reside on this client.</p> <p>The <plan> must be one of the registered archive plans. You can get the list of archive plans with the <code>ArchivePlan names</code> CLI command.</p> <p>The optional <indexroot> argument, if given, will force all files in the archive selection to be indexed under the <indexroot> path.</p>
Return values:	<p>On success: the name of the new resource. Use this name to address the resource in all other methods.</p> <p>On failure: an empty string</p>



Method:	addentry
Syntax:	ArchiveSelection <name> addentry <path> [<key> <value> [<key> <value>]..]
Description:	<p>Adds one new <path> to the archive selection <name>. It expects the absolute path to the file or directory to be archived. The file or directory must be located on the client <client> given at the resource creation time (see the create method).</p> <p>If the passed <path> contains blanks, be sure to enclose it in curly braces: {/some/path with blanks/file}. Furthermore, if the <path> contains { and/or } chars themselves, you must escape them with a backslash \ character.</p> <p>To each path, you can assign arbitrary number of <key> and <value> pairs. Those are saved in the archive index and can be used for searches during restore (see RestoreSelection).</p> <p>Each key allows a string value of unlimited length. If the value contains blanks, it should be enclosed in curly braces. If the value itself contains curly braces, you must escape them with \ character.</p>
Return values:	<p>On success: returns the name of the new ArchiveEntry resource. This name must be used with ArchiveEntry methods to get the status and other meta-information of the entry after the archive operation has been completed. Please see the ArchiveEntry resource description</p> <p>On failure: an empty string</p>



Method: **adddirectory**

Syntax: ArchiveSelection <name> adddirectory <path>
[<key> <value> [<key> <value>]..]

Description: Adds one new directory <path> to the archive selection <name>. It expects the absolute path to the directory to be archived. The directory must be located on the client <client> given at the resource creation time (see the `create` method).

Note that this method will only add the directory node to the archive selection and that only a directory node itself will be archived. If you want to archive both the directory and its contents recursively, use the `ArchiveSelection addentry` method.

See the `addentry` method description for explanation of other method arguments.

Return values: On success: See the `addentry` method description for return values
On failure: an empty string

Method: **size**

Syntax: ArchiveSelection <name> size

Description: Returns the number of entries in the selection object

Return values: On success: number of entries
On failure: an empty string

Method: **destroy**

Syntax: ArchiveSelection <name> destroy

Description: Explicitly destroys the archive selection. The <name> should not be used in any ArchiveSelection commands afterwards

Return values: On success: 0 - destroyed,
1 - not destroyed.
On failure: an empty string



Method:	submit
Syntax:	ArchiveSelection <name> submit [<when>]
Description:	<p>Submits the archive selection for execution. You can optionally override plan execution times by giving the <now> as true (one of "1", "t", "true", "True", "y", "yes", or "Yes").</p> <p>This command implicitly destroys the ArchiveSelection object for the user and transfers the ownership of the internal underlying object to the job scheduler. You should not attempt to use the <name> afterwards.</p>
Return values:	<p>On success: returns the archive job ID. Use this job ID to query the status of the job by using Job resource. Please see the Job resource description for details.</p> <p>On failure: an empty string</p>

RestoreSelection

The restore selection is used to prepare one or more files for the restore operation. You must create new restore selection resource for each new restore session. You can use the resource methods to populate the selection (i.e. add files) and then submit the entire selection for immediate or scheduled execution.

The restore selection is a temporary resource. It does not survive system crashes and server shutdowns, nor it needs to be explicitly destroyed by the caller. It goes out of scope by invoking the "submit" method, which effectively passes the control to the Job manager. The owner of the archive selection resource is thus the PresSTORE system, so the caller needs not (nor it should) perform any other task with the same resource.

Usage:

To use the RestoreSelection, you must first use the `create` method to create new instance. After the creation, you use the `addentry` and or `findentry` methods to fill-in the selection with files to restore. Finally, you must submit the selection for immediate or scheduled execution. After the submission, the resource goes out of scope and should not be used any more.



Method: **create**

Syntax: RestoreSelection create <client> [<relocate>]

Description: Creates new temporary restore selection resource. The resource will be automatically deleted after the associated archive job has been submitted.

The <client> must be one of the registered client computers on the current PresSTORE server. Restored files will be placed on the named client. You can get the list of client computers with the "Client names" CLI command.

The <relocate> overrides default restore location. If this option is given, it must point to a directory on the <client> filesystem. All files will be placed in this directory, instead at their original location. The <relocate> directory will be created, if needed.

Return values: On success: the name of the new resource. Use this name to address the resource in all other methods.

On failure: an empty string

Method: **addentry**

Syntax: RestoreSelection <name> addentry <archiveentry> [<volume>]

Description: Adds a new entry <archiveentry> to the restore selection <name>. The <archiveentry> is a handle to the archived file as returned by the "ArchiveSelection addentry".

The optional <volume> argument, if given, designates the particular media where the entry has been stored (see the [ArchiveEntry volume](#) command for more information).

This is mostly useful when plan which was used to archive the entry is configured for cloning by using two media pools.

Return values: On success: returns path to the file to be restored.

Note: the returned path is not translated to match optional <relocate> argument given at resource creation.

On failure: an empty string



Method: **findentry**

Syntax: RestoreSelection <name> findentry <plan> <expr> [<volume>]

Description: Fills in the restore selection object by searching the archive entries archived with the archive <plan>. The <expr> contains the search expression used to locate records. The <expr> has the following generic format:
<key1> <op1> <val1> && <key2> <op2> <val2> ...

The <key> is the name of the key as passed during archiving of the entry in `ArchiveSelection <name> addentry` or `ArchiveSelection <name> adddirectory` methods.

The <op> is the logical operation applied to the value. The <val> is the value associated with the key. The following logical operations are supported:

- "==" key equals the value
- "*=" key starts with value

Example: `author *= marco && state == italy`

The optional <volume> argument, if given, designates the particular media where the entry has been stored (see the `ArchiveEntry volume` command for more information). This is mostly useful when plan which was used to archive the entry is configured for cloning by using two media pools.

NOTE: Only entries that are located on "known" or "accessible" volumes are reported. If an entry is found in the index but is located on "unaccessible" volume (volume is disabled, not currently mounted in some tape drive or not found in any known media changer), it is not included in the selection.

Return values: On success: number of entries in the selection
On failure: an empty string

Method: **destroy**

Syntax: RestoreSelection <name> destroy

Description: Explicitly destroys the restore selection. The <name> should not be used in any RestoreSelection commands afterwards

Return values: On success: 0 - destroyed, 1 - not destroyed
On failure: an empty string



Method:	submit
Syntax:	RestoreSelection <name> submit [<when>]
Description:	Submits the restore selection for execution. The execution is started immediately, unless the <when> is given. In that case, the execution will be scheduled at the given time. The <when> is the value as returned with the Unix time() function; it is the number of seconds since the Epoch (00:00:00 UTC, 70/1/1).
Return values:	On success: the restore job ID. Use this job ID to query the status of the job by using Job resource. Please see the Job resource description for details. On failure: an empty string



ArchiveEntry

The archive entry represents one archived file. It is an opaque handle which PresSTORE uses to quickly locate the file on the archive media and its metadata in the archive index database.

The archive entry is generated for each file added to the archive selection. Please see the [ArchiveSelection resource](#) description for more details.

Method:	handle
Syntax:	ArchiveEntry handle <client> <path> [<database>]
Description:	<p>Returns the properly formatted archive entry handle which can be used for restoring files archived over the PresSTORE web GUI.</p> <p>The <client> is the name of the PresSTORE client where the <path> resides.</p> <p>The <path> is the absolute platform-native path to a file. No checking is performed on the file. If the passed <path> contains blanks, be sure to enclose it in curly braces: {/some/path with blanks/file}.</p> <p>Furthermore, if the <path> contains { and/or } chars themselves, you must escape them with a backslash \ character.</p> <p>The optional <database> declares the name of the database where the file has been indexed. If omitted, the standard <code>Default-Archive</code> database is used. If no such database could be found in the current PresSTORE configuration, an error is triggered.</p>
Return values:	<p>On success: the handle of the entry</p> <p>On failure: an empty string</p>



Method:	status
Syntax:	ArchiveEntry <name> status
Description:	<p>Returns the status of the archived entry. An archive entry can have number of internal stati, depending on the stage of the archive and/or restore process. Currently, the following stati are supported:</p> <ul style="list-style-type: none">• indexed found in the archive index• unknown not found in the archive index <p>The <code>indexed</code> status means that the entry has been processed (archived) and its meta data may be obtained from the index database.</p> <p>The <code>unknown</code> status means that the entry has not (yet) been found in the index, which is normal for files still waiting to be archived.</p> <p>If the status of an entry returns "unknown" then all of the subsequent entry methods described below will return invalid values.</p>
Return values:	<p>On success: one of the supported stati</p> <p>On failure: an empty string</p>

Method:	volume
Syntax:	ArchiveEntry <name> volume
Description:	<p>Returns the media volume ID where the entry <name> has been archived. An entry can be stored on one or more volumes or even many times on the same volume (see the Volume resource for more information) during the archive operation, depending on the plan configuration.</p>
Return values:	<p>On success: the ID of the volume if the entry was stored on only one volume, or a list of volumes ID's if the entry was stored on multiple volumes</p> <p>On failure: an empty string</p>



Method: **block**

Syntax: ArchiveEntry <name> block

Description: Returns the block numbers on the media volumes where the entry <name> is stored. This is not generally needed by the API user, but is used in PresSTORE internally to quickly locate the entry on the media volume.

Return values: On success: the block number on the volume where the entry is stored. If the entry is stored on more than one volume, a list of block numbers.
On failure: an empty string

Method: **mtime**

Syntax: ArchiveEntry <name> mtime

Description: Returns the list of modification times in Unix time_t format for each instance of the given archive entry.

Return values: On success: list of modification times
On failure: an empty string

Method: **btime**

Syntax: ArchiveEntry <name> btime

Description: Returns the list of backup/archive times in Unix time_t format for each instance of the given archive entry.

Return values: On success: list of backup times
On failure: an empty string

Method: **size**

Syntax: ArchiveEntry <name> size

Description: Returns the list of sizes in bytes for each instance of the given archive entry.

Return values: On success: list of file sizes
On failure: an empty string

Method: **uid (DEPRECATED)**

Syntax: ArchiveEntry <name> uid

Description: Returns the list of Unix uid_t values (user ID) for each instance of the given archive entry.

Return values: On success: list of uid_t values
On failure: an empty string



Method: **gid (DEPRECATED)**
Syntax: ArchiveEntry <name> gid
Description:
Return values: On success: the list of Unix gid_t values (group ID) for each instance of the given archive entry.
On failure: an empty string

Method: **meta**
Syntax: ArchiveEntry <name> meta [<key>]
Description: Returns defined meta-data keys and their values for the given archive entry. If the optional <key> argument is given, it is assumed to be one of the meta columns defined for the particular index database where the archive entry has been indexed.
Return values: On success: with optional <key> argument:
value of the given meta key
without optional <key> argument:
list of all meta keys and their values
On failure: an empty string



Volume

This resource tracks volumes configured for data storage. A volume is an instance of the physical media (tape, digital versatile disk, etc) prepared for use by the PresSTORE server. The preparation of media includes writing of the special label on the beginning of media. By using this label, the PresSTORE server can uniquely identify the media in its volume database.

Method:	names
Syntax:	Volume names
Description:	Returns list of names of all volume resources
Return values:	On success: list of volume names. If no volumes have been configured, returns the string: "<empty>". On failure: an empty string

Method:	label
Syntax:	Volume <name> label [<newvalue>]
Description:	Returns a human-readable description of the volume <name>. If the optional <newvalue> is given, it will set the label to the given value.
Return values:	On success: the volume label On failure: an empty string

Method:	location
Syntax:	Volume <name> location [<newvalue>]
Description:	Returns the physical location of the volume <name>. If the optional <newvalue> is given, it will set the location to the given value.
Return values:	On success: the location string. If the volume location is not set, the command returns the string: "<empty>". On failure: an empty string



Method: **state**

Syntax: Volume <name> state [<newvalue>]

Description: Returns the current state of the volume <name>. The state can be one of:

- Ok
- Suspect

If the optional <newvalue> is given, it will set the state to the given value.

Return values: On success: the volume state
On failure: an empty string

Method: **mode**

Syntax: Volume <name> mode [<newvalue>]

Description: Returns the current mode of the volume <name>. The mode can be one of:

- Appendable
- Closed
- Readonly
- Recyclable
- Full

If the optional <newvalue> is given, it will set the mode to the given value.

Return values: On success: the volume mode
On failure: an empty string

Method: **mediatype**

Syntax: Volume <name> mediatype

Description: Returns the type of media for the volume <name>. This is defined to be one of:

- TAPE
- DISK
- OPTICAL

Return values: On success: the media type
On failure: an empty string



Method: **usedsize**
Syntax: Volume <name> usedsize
Description: Returns the number of kbytes currently written on the volume <name>. If this method returns zero (0) then no data has been written to this volume.
Return values: On success: the number of kbytes written
On failure: an empty string

Method: **maxsize**
Syntax: Volume <name> maxsize
Description: Returns the total number of kbytes which the volume <name> can hold.

This is defined for the mediatype `OPTICAL` and `DISK`. Other types of media, most notably "TAPE" do not have this size defined. If you attempt to get the maxsize of the TAPE media, you will get zero (0) as return value.
Return values: On success: the size in kbytes
On failure: an empty string

Method: **dateused**
Syntax: Volume <name> dateused
Description: Returns the date when the volume is used (for reading or for writing) for the last time. The value returned is the same as the value returned from the Unix `time()` function; it is the number of seconds since the Epoch Date (00:00:00 UTC, 70/1/1).
Return values: On success: the date in seconds since the Epoch
On failure: an empty string

Method: **usage**
Syntax: Volume <name> usage
Description: Returns the current usage of the volume <name>. Currently, following usage options are supported:

- Archive volume must be used for archive jobs
- Backup volume must be used for backup jobs
- Import volume is part of the imported media pool

Return values: On success: the volume usage
On failure: an empty string



Method: **barcode**

Syntax: Volume <name> barcode

Description: Returns the barcode of th volume <name> . If no barcode is present, the string “<empty>” is returned.

This method is introduced in PresSTORE 3.2

Return values: On success: the barcode
On failure: the an empty string

Method: **usecount**

Syntax: Volume <name> usecount

Description: Returns the number of uses for read and/or write operations.

This method is introduced in PresSTORE 3.2

Return values: On success: the number of uses
On failure: an empty string

Jukebox

This resource tracks jukeboxes configured for data storage. Currently you do not have much control over jukeboxes, except getting the list of currently loaded volumes, resetting the jukebox and performing an barcode or mount inventory. Future versions of CLI will allow you to control jukebox resources in a more advanced way.

Method: **names**

Syntax: Jukebox names

Description: Returns list of names of all jukebox resources

Return values: On success: list of jukebox names. If no jukeboxes are configured, the command returns the string: "<empty>".
On failure: an empty string



Method: **volumes**

Syntax: Jukebox <name> volumes

Description: Returns list of all volumes currently loaded in the <name> jukebox. To update the list of the volumes in the jukebox, use the `inventory` method.

Return values: On success: list of volume names.
On failure: an empty string

Method: **reset**

Syntax: Jukebox <name> reset

Description: Performs the hardware jukebox reset, with force emptying all jukebox drives. Use this method with caution since this command will perform an unconditional jukebox reset regardless of any jobs using the jukebox resources.

Return values: On success: true (1)
On failure: an empty string

Method: **inventory**

Syntax: Jukebox <name> inventory [barcode [<startSlot> [<endSlot>]]]

Description: Performs the jukebox <name> inventory, effectively updating the internal volume database.
If the optional barcode argument is specified, it attempts a barcode inventory. If not, a mount inventory of the jukebox is scheduled. If the optional <startSlot> argument is given it is taken as the first slot for the inventory job. Otherwise, the first configured slot of the jukebox is taken.
If the optional <endSlot> argument is given, it is taken as the last slot for the inventory job. Otherwise, the last configured slot of the jukebox is taken.

Return values: On success: the job ID of the scheduled inventory job
On failure: an empty string



Device

This resource tracks tape devices, including single tape drives, tape drives within a jukebox and drives in a virtual jukebox.

Method:	names
Syntax:	Device names
Description:	Returns list of single tape device resources
Return values:	On success: list of device names. If no devices are configured, the command returns the string: "<empty>". On failure: an empty string

Method:	inventory
Syntax:	Device <name> inventory
Description:	Performs the device <name> inventory, effectively updating the internal volume database. Note that this is always a mount inventory, not a barcode inventory. Returns the name of the currently loaded volume
Return values:	On success: the volume name On failure: an empty string



Job

Job resource tracks jobs submitted to PresSTORE server. Information about each submitted job is held indefinitely and can be queried by the user any time. Job resources are generated automatically by the submit methods of the ArchiveSelection and RestoreSelection resources.

Method:	names
Syntax:	Job names
Description:	Returns list of all currently scheduled jobs
Return values:	On success: names of currently scheduled jobs. If no jobs are scheduled, the command returns: "<empty>" On failure: an empty string

Method:	status
Syntax:	Job <name> status
Description:	Returns the status of the job. A job can have number of internal stati, depending on the stage of the archive and/or restore process. Currently, following stati are supported: <ul style="list-style-type: none">• started job is starting (intermediate state)• stopped job is stopping (intermediate state)• unknown ob is not known by the system• scheduled ob is in the queue waiting to be run• running job is running• cancelled ob is cancelled by user• completed ob is completed• terminated job is terminated by server shutdown
Return values:	On success: One of the supported stati On failure: an empty string



Method:	completion
Syntax:	Job <name> completion
Description:	Returns completion code of the completed job. The completion code can be one of: <ul style="list-style-type: none">• success• failure• exception <p>The completion code of "success" means that the job has completed successfully in its entirety. It does not mean that all of the files have been archived and/or restored, though. For info about the particular file, use the "protocol" method.</p> <p>The completion code of "exception" means that parts of the job have failed, but job could have been partially executed ok. This happens for parallel archive/restore operations where one of the job threads may run into error, while others continue to run and finish successfully.</p> <p>The completion code of "error" means that the job has failed in its entirety and none of the files have been processed (archived/restored) correctly.</p>
Return values:	On success: one of the completion codes On failure: an empty string

Method:	report
Syntax:	Job <name> report
Description:	Returns report of the currently running job. The report contains human readable text.
Return values:	On success: the report text On failure: an empty string

Method:	protocol
Syntax:	Job <name> protocol [<archiveentry>]
Description:	Returns completion protocol of the completed job and/or of one of the archived and/or restored file(s) given by the optional <archiveentry> argument. The protocol contains human readable text.
Return values:	On success: the requested protocol On failure: an empty string

Method:



Method: **stop**

Syntax: Job <name> stop

Description: Stops the scheduled job. Only jobs of status `scheduled` can be stopped. An attempt to stop a jobs with a different status will result in error.

Return values: On success: If the job is stopped, returns 1 (true).
If the job could not be stopped (for whatever reason), returns 0 (false)
On failure: an empty string

Method: **cancel**

Syntax: Job <name> cancel

Description: Cancels the running the job. Only jobs of status `running` can be cancelled. An attempt to cancel jobs with a different status will result in error.

Return values: On success: f the job is cancelled, returns 1 (true).
If the job could not be cancelled (for whatever reason), returns 0 (false)
On failure: an empty string

Method: **resourcegroup**

Syntax: Job <name> resourcegroup

Description: Returns the name of the resource group for which this job has been running.

Return values: On success: Name of the resource group (ArchivePlan, SyncPlan, etc).or string "<empty>" if no resource group associated with the job.
On failure: an empty string

Method: **resourcename**

Syntax: Job <name> resourcename

Description: Returns the name of the resource for which this job has been running

Return values: On success: Name of the resource (e.g Default-Backup, Default Archive) or string "<empty>" if no resource group associated with the job.
On failure: an empty string



Method:	xmlticket
Syntax:	Job <name> xmlticket [outfilename]
Description:	Returns completion protocol of the completed job. The protocol contains human readable text embedded in generic XML sections. If the optional <outfilename> argument is given, the output of the command is rerouted to the given file.
Return values:	On success: the requested protocol On failure: an empty string

Pool

This resource tracks volume pools. Volume pools are collection of labeled media which can be used for archive and/or backup tasks.

Method:	names
Syntax:	Pool names
Description:	Lists all configured media pools.
Return values:	On success: list of pool names. If no pools have been configured, the command returns string: "<empty>". On failure: an empty string



Method: **create**

Syntax: Pool create <name> [option value]

Description: Creates one media pool with the name <name>. The <name> of the pool may not include blanks or any special interpunction and/or national characters. If the pool <name> already exists in the PresSTORE configuration an error will be thrown.

Options supported by this command are:

usage one of "Archive" or "Backup"

mediatype one of "OPTICAL", "TAPE" or "DISK"

If no optional arguments are given, the newly created pool will have usage of "Archive" and media type of "TAPE".

Newly created pool will be configured for no parallelism i.e. it will use only one media-device for writing and/or reading from media. If you need to configure the pool for parallelism, please use the PresSTORE Web-GUI.

Example:

```
Pool create MyPool usage Archive mediatype TAPE
```

Return values: On success: the name of the created pool
On failure: an empty string

Method: **usage**

Syntax: Pool <name> usage

Description: Returns either "Archive" or "Backup"

Return values: On success: the usage string
On failure: an empty string

Method: **mediatype**

Syntax: Pool <name> mediatype

Description: eturns one of "OPTICAL", "TAPE" or "DISK" designating the media type of labeled volumes in the pool.

Return values: On success: the media-type string
On failure: an empty string



Method:	volumes
Syntax:	Pool <name> volumes
Description:	Lists all labeled volumes for the given pool
Return values:	On success: list of volume ID's labeled for the named pool. If the pool has no volumes, returns: "<empty>" On failure: an empty string



Example CLI usage

The following examples are made using `nsdchat` utility from the shell script on the PresSTORE server machine. The `nsdchat` utility is invoked in the interactive mode.

```
# cd /usr/local/aw
# bin/nsdchat

% ArchivePlan names
1000

% ArchivePlan 1000 describe
Default archive plan

% ArchiveSelection create localhost 1000
ArchiveSelection.0

% ArchiveSelection ArchiveSelection.0 addentry /usr/local/aw/start-server
Default-Archive#L3Vzci9sb2Nhbc9hdy9zdGFydC1zZXJ2ZXI=

% ArchiveSelection ArchiveSelection.0 submit 1
10190

% Job 10190 status
running

% Job 10190 report
Default-Archive: pool needs new volume -> next check at 13:01:27

% Job 10190 cancel
1

% Job 10190 status
completed

% Job 10190 protocol

No save took place due to early errors!
No volumes found

% exit
```