ARCHIWARE P5

Command Line Interface Manual

Content

Change History

Changes in Version	Changes in methods New methods
5.2.2	srvinfo buildstamp srvinfo hostid srvinfo uptime ArchiveSelection addfile ArchiveSelection addfileabs Client isthin RestoreSelection create
5.3.0	Changed ArchiveSelection addfrom to add folders only without their contents
5.4.3	Pool enabled Pool disabled ArchivePlan incrlevel
5.5.0	nsdchat timeout environment variables ArchiveEntry clippath ArchiveSelection level ArchiveSelection describe RestoreSelection describe SyncSelection onjobactivation (doc correction) Volume inventory (doc correction) Volume location: extended by slot (already in 5.4.4) Workstation name (already in 5.4.4) Job inventory (doc correction)
5.6.2	ArchiveSelection addentry (doc correction) RestoreSelection addfrom (already in 5.5.0) Volume Jobs
5.6.3	RestoreSelection size new ArchiveSelection entries new ArchiveSelection size marked as deprecated
5.6.5	Pool drivecount new Pool create new option blocksize Jukebox volumes new option slotID Jukebox slotcount Jukebox label new Volume dateexpires new
6.0.1	srvinfo home new
6.0.2	RestoreSelection addfromvolume new User Name password new
6.1.0	Job totalkbytes new

Changes in Version	Changes in methods New methods
	Job totalfiles new Job completion (doc correction)
7.0.0	BackupPlan – 21 new commands BackupTask – 11 new commands CalendarEvent – 7 new commands Filter – 12 new commands Volume – 5 new commands Overview

Document revision 09/22, P5 7.0.0

The P5 CLI (Command Language Interface) is a means of accessing the P5 command language, as implemented within the P5 application server.

The CLI allows you to create, query, modify and destroy various P5 resources. A resource is, for example the *client*, *filter*, *backup plan*, *archive plan*, and the like. Resources are tracked in the P5 configuration database.

The CLI can be accessed in several ways, ranging from simple shell-scripts running on the same computer as the P5 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 P5 distribution. The second is the **libchat** library that you can use to link with a C-program.

The *nsdchat* Utility

This is the Unix command-line program for gaining access to the CLI from shell scripts. The `nsdchat` utility is located in the bin/ subdirectory of the P5 installation directory.

The general syntax of the `nsdchat` utility is:

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

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

The relevant option of the `nsdchat` with respect to the CLI is the `-c` option. The `-c` option executes the CLI command with optional arguments on the default P5 server, like for example:

```
nsdchat -c ArchivePlan names
```

The above CLI command lists the names of all the known archive plans located on the default P5 server running on the local computer.

By using the "-s" option of the `nsdchat` utility you can specify a P5 server other 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 TCP sockets. On Windows, TCP sockets must be specified.

Named-pipes can be used only when the `nsdchat` utility and the P5 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	host name or IP address of the P5 host
`<port>`	required	port number of the P5 socket server

Examples:

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

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
`<homedir>`	required	P5 installation directory
`<server>`	required	currently, only `lexxsrv` is allowed

Note: On Windows, this connection type is not supported, as Windows does not support named pipes. Instead, please use the TCP socket connection described above.

Example

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

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 user name (by running two `nsdchat` programs or two programs linked with the `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 a 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 mode 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 P5 server. Additionally, you can also make the `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 P5 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.

Environment Variables:

The `nsdchat` utility communicates with the P5 server which in turn executes the given command. For that connection, timeout values are used in `nsdchat`, which can be influenced by environment variables. The following table shows the variables with their default values:

Variable	Default (sec)	Description
NSDCHAT_CONN_TOUT	300	Timeout to connect to the P5 server port
NSDCHAT_LOGIN_TOUT	120	Timeout for the check of username/password
NSDCHAT_CMD_TOUT	3600	Timeout for the CLI command completion
NSDCHAT_COMM_TOUT	120	Timeout for the next byte on the comm channel

Example

```
export NSDCHAT_CMD_TOUT=7200
bin/nsdchat -c <CLI COMMAND>
```

This allows for two hours to complete of the given CLI command.

Special considerations on Windows

The commands and the command syntax in this document have been written for Unix-like systems. On Windows, the commands work the same way, but there are some minor differences regarding the parameters passed to the commands:

Paths

All paths in P5 start with a slash sign "/" and use slashes as path delimiters. When specifying Windows paths, please use this syntax:

/C/my/folder

instead of

C:\my\folder

Also please note that each path should be given as an absolute path. Relative paths are always relative to the P5 home folder. So the path `./myfolder` would resolve to `/C/Program Files/ARCHIWARE/Data_Lifecycle_Management_Suite/my_folder.`

Arguments

Some arguments must be passed as a single argument through nsdchat to P5, for instance when passing paths containing a blank. The Windows CMD shell, same as the sh-shell on Linux, will treat a blank as a separator and pass two arguments. In order to allow P5 to regard that as a single argument, it is required to enclose the argument in curly braces. For instance the path

`C:\MY Data\my folder`

must be specified in P5 syntax and in curly braces as

`{/C/MY Data/my folder}`

in addition, it is required to enclose the string in quotes like

`"{/C/MY Data/my folder}"`

To ensure the CMD shell passes it as a single argument.

Note that argument passing may depend on what program is calling a command and interpreting the parameters. So the call to nsdchat may under special conditions behave differently when typed in on a command shell or when called from within another program, specially regarding the parameter separation.

Calling nsdchat

Windows does not support named pipes in the file system. Due to that limitation, nsdchat on Windows must always use the tcp based communication to P5. The call thus must always contain the option "`-s awsock: ...`"

See above for the parameter details.

The *libchat* Library

This library is provided on request and is used to be linked with your C-program to gain access to the CLI. The library exposes a very simple API for logging-in to the P5 server, sending commands and processing results. The library is available for all supported P5 platforms in both static and shared-object form.

The library offers the same functionality as the `nsdchat` utility with one notable exception: the library allows the handling of events. P5 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 that will be invoked for each event received on the communication link.

Environmental Variables

The following Environmental Variables are generated by P5:

Variables set on the P5 server:

`AWPST_CLN_HOST`	name of the P5 client host
`AWPST_CLN_PORT`	port of the P5 client
`AWPST_CLN_PCLI`	port for the client CLI communication
`AWPST_CLN_HOME`	installation home directory

Variables set on the P5 client:

`AWPST_SRV_HOST`	name of the P5 server host
`AWPST_SRV_PORT`	port of the P5 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 P5 when processing various jobs.

P5 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 basic 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 a single mandatory argument, which is either an existing resource name, or one of the general or resource specific sub-commands, as described below. Both forms accept a variable number of `arg/value` pairs.

All CLI commands return an empty result (do not return anything) in case of an error. To find out the real cause of the error (display the error message) you can use the `geterror` CLI command. In addition to resource commands, there are other, resource-independent commands that operate on the global level.

The CLI is built on top of the Tcl extension language. It understands all Tcl control structures, so you can write full-fledged Tcl programs. The CLI interpreter runs in the 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 an empty result string from any CLI command to receive an explanation for the encountered error.

srvinfo

This command returns information about the current P5 server.

Method:	**buildstamp**
Syntax:	srvinfo buildstamp
Description:	Returns the build time-stamp of the P5 release
Return values:	The build time-stamp

Method:	**address**
Syntax:	srvinfo address
Description:	Returns the IP address of the P5 host
Return values:	The IP address in standard dot notation

Method:	**home**
Syntax:	srvinfo home
Description:	Returns the P5 home directory, i.e. the path where P5 is installed
Return values:	The home directory

Method:	**hostid**
Syntax:	srvinfo hostid
Description:	Returns the host ID of the P5 host (as shown in the about box)
Return values:	The host ID

Method:	**hostname**
Syntax:	srvinfo hostname
Description:	Returns the host name of the P5 host
Return values:	The host name as returned with the `hostname` shell command

Method:	**lexxvers**
Syntax:	srvinfo lexxvers
Description:	Returns the P5 application version
Return values:	The application version string as X.Y.Z number

Method:	**platform**
Syntax:	srvinfo platform
Description:	Returns the OS platform of the P5 host
Return values:	One of: linux, solaris, windows or macosx

Method:	**port**
Syntax:	srvinfo port
Description:	Returns the TCP port of the P5 server
Return values:	The TCP port number

Method:	**server**
Syntax:	srvinfo server
Description:	Returns the name of the P5 server. Currently there is only one server assigned: lexxsrv.
Return values:	The server name

Method:	**uptime**
Syntax:	srvinfo uptime
Description:	Returns the time in seconds since the P5 server was started
Return values:	The uptime in seconds

Method:	**version**
Syntax:	srvinfo version
Description:	Returns the version of the P5 application server.
Return values:	The application server version string as X.Y number

License-Related Commands

License Information

The returned resource names are internal names of license components that are combined to form a product license.
A product license, like for instance a *Backup Module* AWB100, consists of

- 1 BackupPlan: the *Backup* functionality
- 1 Client: a *Server Agent*
- 1 Device: a *Media Tape License* for a single *Tape Drive*

The set of internal resources does not reflect the exact number or type of installed licenses, it gives a summary of installed license resources.

Method:	**resources**
Syntax:	License resources
Description:	Returns the list of names of all License resources
Return values:	On success: the list of names On failure: an empty string

Method:	**free**
Syntax:	License <resource> free
Description:	For the given resource, returns whether there are free licenses available.
Return values:	On success: the string "-1" for unlimited free licenses the string "0" for no free license or a positive count for the number of free licenses On failure: an empty string
Note:	Trial licenses and license resources that are not countable will return the string "-1", if available.

Account-Related Commands

Starting with version 6, P5 uses internal passwords and password authentication during user login. In order to change a password in the GUI, the user preferences must be used. In addition, the CLI allows to change a user password, provided the current password for that user is known.

Method:	**password**
Syntax:	User <name> password <newpassword> <oldpassword>
Description:	Sets a new user password for user account <name> The password to be set must be given as <newpassword>, the current password of that account must be given as <oldpassword> .
Return values:	On success: “1” (new password is set) On wrong password: “0” (new password is not set) On failure: an empty string

Backup2Go-Related Commands

Backup2Go Templates / Workstation Groups

Queries *Backup2Go* templates configured on the *Backup2Go Server* and queries and controls their parameters. These commands are to be executed on the *Backup2Go* server.

Status and Information

Method:	**names**
Syntax:	Backup2Go names
Description:	Returns the list of names of all the Backup2Go templates
Return values:	On success: the list of names On failure: an empty string

Method:	**describe**
Syntax:	Backup2Go <name> describe
Description:	Returns a human-readable description of the template <name>. If the template does not have a description assigned, the command returns the string "*<empty>"*
Return values:	On success: the workstation description On failure: an empty string

Method:	**disabled**
Syntax:	Backup2Go <name> disabled
Description:	Queries Backup2Go template `Disabled` status
Return values:	On success: the string “1" (disabled) or "0" (not disabled) On failure: an empty string

Method:	**enabled**
Syntax:	Backup2Go <name> enabled
Description:	Queries the template `Enabled` status.
Return values:	On success: the string “1" (enabled) or "0" (not enabled) On failure: an empty string

Method:	**disable**
Syntax:	Backup2Go <name> disable
Description:	Sets the template to the `Disabled` state
Return values:	On success: the string "0" On failure: an empty string

Method:	**enable**
Syntax:	Backup2Go <name> enable
Description:	Sets the template to the `Enabled` state
Return values:	On success: the string "1" On failure: an empty string

Method:	**cleanup**
Syntax:	Backup2Go cleanup [snapshots] [trashes]
Description:	Purges selected Backup2Go areas. It does not wait for the completion of the command. Instead, it schedules an internally queued job and does the work in the background.
Return values:	On success: the string "*ok*" On failure: an empty string

Method:	**maxrunning**
Syntax:	Backup2Go <name> maxrunning [<count>]
Description:	Set up or report the maximum number of active workstations for the given template.
Return values:	On success: the number of active workstations, the string "-1" for unlimited On failure: an empty string

Workstation

Queries *Backup2Go* workstation resources configured on the *Backup2Go Server* and queries and controls their parameters. These commands are to be executed on the *Backup2Go* server.

A P5 workstation is the computer running the P5 client software in a *Backup2Go* infrastructure. To configure and maintain workstation resources, use the standard system-administrator account in the P5 Web GUI

Status and Information

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

Method:	**describe**
Syntax:	Workstation <name> describe
Description:	Returns a human-readable description of the workstation <name>. If the workstation does not have a description assigned, the command returns the string "*<empty>"*
Return values:	On success: the workstation description On failure: an empty string

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

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

Method:	**hostid**
Syntax:	Workstation <name> hostid
Description:	Returns the configured P5 machine-ID of the workstation <name>.
Return values:	On success: the workstation's machine ID On failure: an empty string

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

Method:	**lastend**
Syntax:	Workstation<name> lastend
Description:	Returns the absolute time in seconds (Posix time) of the successful end of the last backup operation for the workstation <name>. This time may be older then the time returned by the *lastbegin* method indicating an incomplete (interrupted) backup.
Return values:	On success: the time in seconds (Posix time) On failure: an empty string

Method:	**lasterror**
Syntax:	Workstation <name> lasterror
Description:	Returns the error message that resulted from the last backup run for the workstation <name>. The string "*<empty>"* is returned in case there is no last error.
Return values:	On success: the error message or the string "*<empty>"* On failure: an empty string

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

Method:	**peerip**
Syntax:	Workstation <name> peerip
Description:	Returns the last known IP of the workstation <name>. If the workstation does not have an IP recorded 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:	**snapshots**
Syntax:	Workstation <name> snapshots [<since>]
Description:	Returns a list of snapshots maintained for the given workstation. The optional <since> argument may be given in seconds (Posix time) to address only snapshots since that date. Otherwise all known snapshots are returned.
Return values:	On success: a list of snapshots IDs On failure: an empty string

Method:	**snapsize**
Syntax:	Workstation <name> snapsize [<snapshotId>]
Description:	Returns the allocated size in KBytes of data maintained for the named workstation. On link based snapshots, one or multiple <snapshotId> arguments (as returned by the **snapshots** method) can be given. The return value is then the allocated size for the current and all optional given snapshots summed up. On native snapshots (ZFS, BTRFS), this method accepts one or none <snapshotId> as parameter. If a snapshot ID is given, the logical size of that snapshot is returned, otherwise the size of the current state is returned. The return value does not reflect the required disk space of native snapshots. All returned sizes are in Kbyte. Note that this may be a lengthy operation, depending on the number of files and snapshots.
Return values:	On success: the number of KBytes On failure: an empty string

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

Method:	**retaintime**
Syntax:	Workstation <name> retaintime
Description:	Returns the retention time setting for workstation snapshots.
Return values:	On success: the retention time in seconds On failure: an empty string

Method:	**template**
Syntax:	Workstation <name> template
Description:	Returns the template ID for workstation <name>.
Return values:	On success: the template ID On failure: an empty string

Method:	**configure**
Syntax:	Workstation configure <hostname> <port> <username> <password> [<template>]
Description:	Run this command on the P5 Backup2Go Server. Using the passed connection parameters <hostname> and <port>, tries 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 in to the server, the workstation will be seeded with a unique token, shared by the workstation and the server. This eliminates the 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.
Return values:	On success: a positive integer as a string (the name of the new local workstation) On failure: the string "-3": the template could not be set the string "-2": a wrong user name/password is given the string "-1": there is a network connection problem (bad address and/or port)

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

To be executed on the Workstation

This command must be executed on the Backup2Go workstation.

Method:	**name**
Syntax:	Workstation name
Description:	Returns the Workstation ID of the workstation where the command is executed **Note:** Unlike all the other workstation commands, this command must be called on the Workstation
Return values:	On success: the ID or the string "*unknown*" On failure: an empty string

Server

Queries P5 *Backup2Go* server resources configured on the *Backup2Go* workstation and their parameters. A P5 server is the computer running the P5 server software and providing backup services to P5 workstation computers. These commands are to be executed on the *Backup2Go* workstation.

General

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

Method:	**create**
Syntax:	Server create
Description:	Creates a new server resource
Return values:	On success: the 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: the string "1" if deleted or "0" if not On failure: an empty string

Status and Information

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

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

Method:	**lastbegin**
Syntax:	Server <name> lastbegin
Description:	Returns the absolute time in seconds (Posix time) of the beginning of the last backup operation on the server <name>
Return values:	On success: the time in seconds (Posix time) On failure: an empty string

Method:	**lastend**
Syntax:	Server <name> lastend
Description:	Returns the absolute time in seconds (Posix time) of the successful end of the last backup operation on the server <name>. This time may be older then the time returned by the *lastbegin* method, indicating an incomplete (interrupted) backup.
Return values:	On success: the time in seconds (Posix time) On failure: an empty string

Method:	**nextrun**
Syntax:	Server <name> nextrun
Description:	Returns absolute time in seconds (Posix time) of the beginning of the next scheduled backup operation to the server <name>. It will return the string "0" if no scheduled backup is present.
Return values:	On success: the time in seconds (Posix time) On failure: an empty string

Method:	**template**
Syntax:	Server <name> template
Description:	Returns the server-side template ID used for the backup operation to the server <name>. If no template ID is assigned, it will return the string "*<empty>*".
Return values:	On success: the template ID On failure: an empty string

Method:	**configure**
Syntax:	Server configure <host> <port> <user name> <password> [<template>]
Description:	Creates new (or reuses existing) server resource and configures the required connection parameter in a single call. If the optional <template> argument is set, it forces the selection of the given template on the server, otherwise the default template is used.
Return values:	On success: name/ID of the created server resource On failure: a negative integer as a string: "-1": Network connection problem (bad host or port) "-2": Wrong user name or password (log in denied) "-3": The template cannot be set (it is disabled or cannot be found)

Method:	**cputhrottle**
Syntax:	Server <name> cputhrottle [<value>]
Description:	If no additional arguments specified, returns the workstation CPU throttle in percent (0% - 100%). 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:	**hostname**
Syntax:	Server <name> hostname [<value>]
Description:	If no additional arguments are specified, returns the host name or IP address of the server. Otherwise it stores the given argument as the new host name.
Return values:	On success: the host name On failure: an empty string

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

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

Method:	**dataencryption**
Syntax:	Server <name> dataencryption [<value>]
Description:	If no additional arguments are specified, returns the boolean corresponding string "0" or "1" 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 it stores the given argument as the new flag value.
Return values:	On success: the boolean corresponding string "0" or "1" On failure: an empty string

Method:	**netencryption**
Syntax:	Server <name> netencryption [<value>]
Description:	If no additional arguments are specified, returns the boolean corresponding string "1" or "0" depending whether the workstation will encrypt the network traffic targeted to this server (1) or not (0). Otherwise it stores the given argument as the new flag value.
Return values:	On success: the boolean corresponding string "0" or "1" On failure: an empty string

Method:	**netthrottle / throttle**
Syntax:	Server <name> netthrottle [<value>] Server <name> throttle [<value>]
Description:	If no additional arguments are specified, returns the bandwidth throttle of the communication link used to talk to this server in percents (0% - 100%). Otherwise it stores the given argument as the new throttle value.
Return values:	On success: the throttle value in percent On failure: an empty string

Method:	**password**
Syntax:	Server <name> password <value>
Description:	Stores the given argument as the new password.
Return values:	On success: the password On failure: an empty string

Method:	**pathlist**
Syntax:	Server <name> pathlist [<value>]
Description:	If no additional arguments are specified, returns the list of paths configured for the backup operation. The 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 it stores the given argument as the new list of paths. Each path in the list must be delimited from the next 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:	**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 server response. If the argument is omitted, the timeout defaults to 600 seconds (10 minutes).
Return values:	The string: "-2" wrong user name or password "-1" network connection problem "0" reserved for future use "1" ping ok

Method:	**port**
Syntax:	Server <name> port [<value>]
Description:	If no additional arguments are specified, returns the TCP port number of the server. Otherwise it stores the given argument as the new port number.
Return values:	On success: the port number On failure: an empty string

Method:	**reschedule**
Syntax:	Server <name> reschedule [<value>]
Description:	If no additional arguments are specified, returns the number of hours to re-schedule the backup job after regular completion. Note that jobs that do not complete regularly are immediately automatically rescheduled. Otherwise it stores the given argument as the new number of hours.
Return values:	On success: the number of hours On failure: an empty string

Method:	**submit/start**
Syntax:	Server <name> submit [<now>] Server <name> start [<now>]
Description:	Submits the workstation backup job for execution to the server <name>. You can optionally override plan execution times by using the verbatim string *now* or the integer value zero for the <now> argument. The returned job ID can be used 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:	**useevents**
Syntax:	Server <name> useevents [<value>]
Description:	If no additional arguments are specified, returns the boolean corresponding string "0" or "1", depending on whether the workstation will use the file system events facility when gathering files of this server (1) to store or will use a linear file system walk (0). Otherwise it stores the given argument as the new value.
Return values:	On success: the boolean corresponding string "0" or "1" On failure: an empty string

Method:	**usecompression**
Syntax:	Server <name> usecompression [<value>]
Description:	If no additional arguments are specified, returns the boolean corresponding string "0" or "1" depending whether the workstation will compress the network traffic targeted to this server (1) or not (0). Otherwise it stores the given argument as the new flag value.
Return values:	On success: the boolean corresponding string "0" or "1" On failure: an empty string

Method:	**username**
Syntax:	Server <name> username [<value>]
Description:	If no additional arguments are specified, returns the name of the user to use for authentication on the current server. Otherwise it stores the given argument as the new user name.
Return values:	On success: the user name On failure: an empty string

Plan- and Client-Related Commands

ArchivePlan

Manages P5 archive plan(s) and their parameters. Archive plans are used to group various parameters of the archive operation, like the selected index database, the pool of media, a time schedule and various other details. The P5 administrator defines archive plans according to the custom site policies. A user who wishes to archive files must select one of the predefined archive plans.

In the current version of the CLI, you only have limited write access to archive plans. You can modify some configuration details of existing plans and you can create new archive plans. If you need full control of ArchivePlan resources, please use the P5 Web GUI.

Status and Information

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

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

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

Method:	**incrlevel**
Syntax:	ArchivePlan <name> incrlevel
Description:	Queries the plan `incremental` status
Return values:	On success: the string "1" (plan is incremental) or "0" (plan runs full) On failure: an empty string

Control Commands

Method:	**autostart**
Syntax:	ArchivePlan <name> autostart
Description:	Returns the autostart setting for the Archive plan <name>. If the Archive plan is set to autostart, the returned value is “1”, otherwise it is “0”.
Return values:	On success: the string "1" (plan is set to autostart) the string “0” (plan is not set to autostart) On failure: an empty string

Method:	**create**
Syntax:	ArchivePlan create <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: the name of the newly created plan On failure: an empty string

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

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

Method:	**deletefiles**
Syntax:	ArchivePlan <name> deletefiles [<value>]
Description:	Returns or sets the option to delete files after successfully completing the archive job. If optional <value> argument is omitted, returns the current setting. If <value> is given (as “true”, “yes” or “1”), enables this option. To also delete the folder structure, use the *deleteall* command.
Return values:	On success: the string "1" (the plan is set to delete files) the string “0” (the plan is set not to delete files) On failure: an empty string

Method:	**deleteall**
Syntax:	ArchivePlan <name> deleteall [<value>]
Description:	Returns or sets the option to delete both files and folders after successfully completing archive plan job. If optional <value> argument is omitted, returns the current setting. If <value> is given (as “true”, “yes” or “1”), enables this option.
Return values:	On success: the string "1" (the plan is set to delete files and folders) the string “0” (the plan is set to not delete anything) On failure: an empty string

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

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

Method:	**pool**
Syntax:	ArchivePlan <name> pool [<value>]
Description:	Returns the name of the media pool associated with the archive plan <name>. If the optional <value> argument is not given, the name of the currently configured pool will be returned. If the optional <value> argument is given it will be taken as the name of an existing media pool, and the plan <name> will be configured to use the given pool. If the referenced media pool is not configured, an error will be thrown. Also, if the referenced media pool is not set up for archive operation, an error will be thrown. You can use the `Pool` resource commands to inspect and/or create media pools. Note that ArchivePlan must have the media pool set. Otherwise, the archive job configured to use this plan will fail.
Return values:	On success: the name of the primary media pool. If not configured, it returns the string "<*empty*>" On failure: an empty string

Method:	**run**
Syntax:	ArchivePlan <name> run [-delete 1]
Description:	Runs the archive plan immediately with an optional delete pass on the target directory/ies. Note: 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: the string "1" (the plan was successfully removed) the string "0" (the plan was not removed or is running) On failure: an empty string

Method:	**submit / start**
Syntax:	ArchivePlan <name> submit [<now>] ArchivePlan <name> start [<now>]
Description:	Submits the archive plan for execution. You can optionally override plan execution times by using the verbatim string *now* or the integer value zero for the <now> argument. The returned job ID can be used to query the status of the job by using the `Job` resource. Please see the `Job resource` description for more details. Note: In order to run an Archive plan, an archive event must be selected. The *start* method thus selects the next planned archive event to start the archive plan.
Return values:	On success: the archive job ID 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.
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 On failure: an empty string

BackupPlan

Queries P5 backup plans and their associated parameters. Backup plans are used to group various parameters of the backup operation, like the pool of media, time schedules and other details. The P5 administrator defines backup plans according to the custom site policies.

General commands

Method:	**create**
Syntax:	BackupPlan create <description>
Description:	Creates a new Backup plan with the given description. The description must not be empty and must be unique among existing backup plans.
Return values:	On success: new backup plan *name* On failure: an empty string

Method:	**delete**
Syntax:	BackupPlan <name> delete
Description:	Deletes the backup plan
Return values:	On success: *an empty string* On failure: an empty string

Method:	**addtask**
Syntax:	BackupPlan <name> addtask
Description:	Creates a new Backup task and adds it to the plan. For details, please see *Backup Task* section.
Return values:	On success: new Backup task *name* On failure: an empty string

Method:	**deletetask**
Syntax:	BackupPlan <name> deletetask <task name>
Description:	Deletes the given Backup task from the plan. For details, please see *Backup Task* section.
Return values:	On success: the string “1” On failure: an empty string

Method:	**newevent**
Syntax:	BackupPlan <name> newevent
Description:	Creates a new Backup event and adds it to the plan. For details, please see *Calendar Event* section.
Return values:	On success: new Backup task *name* On failure: an empty string

Status and Information

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

Method:	**describe**
Syntax:	BackupPlan <name> describe
Description:	Returns a human-readable description for the <name> plan. The <name> is one of the elements returned by the *names* method. If the element does not have a description assigned, the command returns the 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:	**disabled**
Syntax:	BackupPlan <name> disabled
Description:	Queries the `Disabled` status
Return values:	On success: the string "1" (the plan is disabled) or "0" (not disabled) On failure: an empty string

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

Method:	**tasknames**
Syntax:	BackupPlan <name> tasknames
Description:	Returns a list of Backup tasks associated with this Backup plan
Return values:	On success: list of task names or the string “<empty>” On failure: an empty string

Method:	**eventnames**
Syntax:	BackupPlan <name> eventnames
Description:	Returns a list of Backup events associated with this Backup plan
Return values:	On success: list of event names or the string “<empty>” On failure: an empty string

Control Commands

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

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

Method:	**retention**
Syntax:	BackupPlan <name> retention [value]
Description:	If *value* is not specified, returns the backup retention period measured in days. Otherwise sets the retention to the given value.
Return values:	On success: retention period in days On failure: an empty string

Method:	**loglevel**
Syntax:	BackupPlan <name> loglevel [value]
Description:	If *value* is not specified, returns the configured log level where *0=errors only; 1=erros and file operations*. Otherwise sets the loglevel to the given value.
Return values:	On success: log level On failure: an empty string

Method:	**stophours**
Syntax:	BackupPlan <name> stophours [value]
Description:	If *value* is not specified, returns the configured number of hours after which the backup will be stopped. Otherwise sets this number of hours to the given value.
Return values:	On success: number of hours On failure: an empty string

Method:	**progressive**
Syntax:	BackupPlan <name> progressive [value]
Description:	Boolean flag to set (if value is given) or get the usage of progressive backup strategy.
Return values:	On success: 1 or 0 On failure: an empty string

Method:	**canceltime**
Syntax:	BackupPlan <name> canceltime [value]
Description:	If *value* is not specified, returns the configured number of hours after which the backup will be cancelled if the necessary volumes do not become available. Otherwise sets this number of hours to the given value. Value of 0 will cause backups to wait indefinitely.
Return values:	On success: number of hours On failure: an empty string

Method:	**startnewvol**
Syntax:	BackupPlan <name> startnewvol [value]
Description:	Boolean flag to set (if value is given) or get the backup media usage policy, where 1 = start new volume and 0 = append to the existing media.
Return values:	On success: string “1” or “0” On failure: an empty string

Method:	**dryrun**
Syntax:	BackupPlan <name> dryrun <event> [task]
Description:	Starts a dry run job for the Backup plan. The <event> is a Backup event name (id) as returned by the *BackupPlan <name> eventnames* command. If the optional argument [task] is not given or if it is given as a string value “ALL”, the job will use all configured Backup tasks.
Return values:	On success: list of job names On failure: an empty string

Method:	**startnow**
Syntax:	BackupPlan <name> startnow <event> <task> [sizelimit]
Description:	Starts a backup job for the Backup plan. The <event> is a Backup event name (id) as returned by the *BackupPlan <name> eventnames* command. The <task> is a Backup task name and it may given as a string value “ALL” to use all configured Backup tasks. If the optional argument [sizelimit] is given, backup will be limited to this number of bytes.
Return values:	On success: list of job names On failure: an empty string

BackupTask

Queries P5 backup tasks associated with backup plans. In order to get a task's name, please call the *BackupPlan tasknames* command first.

Status and Information

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

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

Control Commands

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

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

Method:	**client**
Syntax:	BackupTask <name> client [value]
Description:	If *value* is not specified, returns the configured client name. Otherwise sets the client to the given value. Valid client names can be queried by *Client names* command.
Return values:	On success: client name On failure: an empty string

Method:	**dirlist**
Syntax:	BackupTask <name> dirlist [value]
Description:	If *value* is not specified, returns the list of configured directory paths. Otherwise sets this list to the given value.
Return values:	On success: list of paths On failure: an empty string

Method:	**crossmounts**
Syntax:	BackupTask <name> crossmounts [value]
Description:	Flag determining whether backups should follow links to external mount points (1) or remain within the single file system (0).
Return values:	On success: string “1” or “0” On failure: an empty string

Method:	**prescript**
Syntax:	BackupTask <name> prescript [value]
Description:	Full path to a script to be executed before backup. Can also be run on a specified client by adding the client name (e.g. *saturn:/usr/local/scripts/db-stop.sh*)
Return values:	On success: script path On failure: an empty string

Method:	**postscript**
Syntax:	BackupTask <name> postscript [value]
Description:	Full path to a script to be executed after backup. Can also be run on a specified client by adding the client name (e.g. *saturn:/usr/local/scripts/db-restart.sh*)
Return values:	On success: number of hours On failure: an empty string

Method:	**runpostonerr**
Syntax:	BackupTask <name> runpostonerr [value]
Description:	Set/get the boolean flag determining whether the postscript should be run even if a backup fails. 1 = run always; 0 = do not run if backup fails.
Return values:	On success: string “1” or “0” On failure: an empty string

Method:	**filecheck**
Syntax:	BackupTask <name> filecheck [value]
Description:	Set/get the array of filters to use when determining if a file needs to be backed up. Possible elements are *'mtime ctime move rtime size*'. If not set, the default value of '*mtime ctime move*' is used.
Return values:	On success: list of filters On failure: an empty string

Method:	**filter**
Syntax:	BackupTask <name> filter [filter name]
Description:	Set/get the file filter for this backup task. [Filter name] is one of the names are returned by the ‘Filter names’ command.
Return values:	On success: list of filters On failure: an empty string

SyncPlan

Queries P5 synchronize plans and their parameters. Sync plans are used to group various parameters of the synchronize operation, like time schedules and various other details. P5 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 P5 Web GUI by the system administrator.

Status and Information

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

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

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

Method:	**sourcehost**
Syntax:	SyncPlan <name> sourcehost
Description:	Returns the name of the client where the source data is located.
Return values:	On success: the name of the client On failure: an empty string

Method:	**sourcepath**
Syntax:	SyncPlan <name> sourcepath [newpath]
Description:	If no optional argment *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: the 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: the 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: the path to the directory 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 the *stop* method)
Return values:	On success: the string "1" (the plan was successfully canceled) the string "0" (the plan was not canceled or running) On failure: an empty string

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

Method:	**enable**
Syntax:	SyncPlan <name> enable
Description:	Sets the plan to the `Enabled` state
Return values:	On success: the string "1" 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. Note: In order to run a Synchronize plan, a Synchronize event must be selected. The *start* method implicitly selects the next planned event to start the backup plan.
Return values:	On success: 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:	**start / submit**
Syntax:	SyncPlan <name> submit [<now>] SyncPlan <name> start [<now>]
Description:	Submits the sync plan for execution. You can optionally override plan execution times by using the verbatim string *now* or the integer value zero for the <now> argument. Plan must be configured for auto-start since CLI just overrides the scheduled starting time. This command cannot be used to start a plan which is not set to auto-start or which does not have any events configured.
Return values:	On success: 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: the string "1" (the plan was successfully removed) the string "0" (the plan was not removed or running) On failure: an empty string

SyncSelection / Temporary Syncplan

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 does it need 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 P5 system, so the caller does not need (nor should) perform any other task with the same resource.

Usage:

To use the SyncSelection resource, you must first use the *create* method to create a new instance. Having created an instance, use the *adddirectory* method to fill-in the selection with directories to synchronize. Finally, submit the selection for immediate or scheduled execution. After 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. Use this name to address the resource in all the 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 On failure: an empty string

Method:	**addrecursive**
Syntax:	SyncSelection <name> addrecursive <path>
Description:	Adds a single new directory <path> to the sync selection <name> and recurses into the subfolders of that directory. 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:	**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: the string "0" (destroyed) the string "1" (not destroyed) On failure: an empty string

Method:	**onjobactivation**
Syntax:	**SyncSelection <name> onjobactivation** [<command>]

Description:	Registers the <command> to be executed just before the job is started by the *submit* method. The command itself can be any valid OS command plus variable number of arguments.
	The very first argument of the command (the program itself) can be prepended with the name of the P5 client where the command is to be executed on. If omitted, the command will be executed on the client which the SyncSelecttion object is created for.
	
	Examples:
	`SyncSelection SyncSelection.0 onjobactivation "mickey:/var/myscript arg"`
	will execute /var//myscript on the client "mickey" regardless what client the SyncSelection is created for. The program will be passed one argument: arg.
	`SyncSelection SyncSelection.0 onjobactivation "/var/scripts/myscript"`
	will execute /var/scripts/myscript on the client the SyncSelection is created for.
	`SyncSelection SyncSelection.0 onjobactivation "localhost:/var/scripts/myscript"`
	will execute /var/scripts/myscript on the P5 server.

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

Method:	**onjobcompletion**
Syntax:	SyncSelection <name> onjobcompletion [<value>]
Description:	Registers the <command> to be executed immediately after the job created by the *submit* method is completed. See *onjobactivation* for further information.
Return values:	On success: the command string On failure: an empty string

Method:	**submit**
Syntax:	SyncSelection <name> submit [<now>]
Description:	Submits the sync selection for execution. You can optionally override plan execution times by giving the <now> as one of the strings "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: 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

Client

Queries configured P5 client resources and their parameters. A P5 client is the computer running the P5 client software. A P5 server is the computer running the P5 server software. A server can archive, backup, restore and synchronize files to and from any registered client.

In the current version of the CLI, there is only read access to client data. You can't modify any of the existing client resources nor can you create new or delete existing clients. To configure and maintain client resources, use the standard system administrator account in the P5 Web GUI.

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

Method:	**describe**
Syntax:	Client <name> describe
Description:	Returns a human-readable description of the client <name>. If the client does not have a 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 the host name (or IP address) of the client <name>
Return values:	On success: the host name or IP address On failure: an empty string

Method:	**isthin**
Syntax:	Client <name> isthin
Description:	Returns true in case the client is of type *Workstation* (as opposed to type *Server*)
Return values:	On success: the string “1” if the client type is *Workstation* the string “0” otherwise 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:	The string: "-4" wrong client version "-3" the client is disabled "-2" wrong user name/password "-1" network connection problem "0" (reserved for future use) "1" ping OK

CalendarEvent

Various operations in P5 are scheduled to run automatically at certain points in time. This scheduling is governed by a calendar comprising one or more *Events*, each one with its own recurrence and other job related settings.

Time description string

Event frequency and exceptions use a specially formatted string to represent their scheduling rules. String is formatted as: *<type> <count> [subset].* There are three recurrence types: Daily, Weekly and Monthly:

Daily:

- <type> = “D”
- <count> = repeat every <count> days
- example: “D 3” = run every three days

Weekly:

- <type> = “W”
- <count> = repeat every <count> weeks
- <days> = run on these days (1=Monday, 2=Tuesday… 7=Sunday)
- example: “W 2 1 2 3 4 5” = run every second week on Mon – Fri

Monthly:

- <type> = “M”
- <count> = repeat every <count> months
- <week> = run in week number <week> (1=first, 2=second...5=last)
- <day> = run on this day (single value, 1=Monday...7=Sunday)
- example: “M 1 3 7” = run every month in the thirds week on Sunday

Backup related events

Method:	**deleteevent**
Syntax:	CalendarEvent <name> deleteevent
Description:	Deletes this event and removes it from the associated Backup plan.
Return values:	On success: string “1” On failure: an empty string
Method:	**pool**
Syntax:	CalendarEvent <name> pool [value]
Description:	Set/get the backup pool name for this event.
Return values:	On success: pool name On failure: an empty string

Method:	**level**
Syntax:	CalendarEvent <name> level [value]
Description:	Set/get the backup level for this event. Possible values are 1 = full backup; 0 = increment; -1 = synthetic. NOTE: Always use *level=1* if the target pool type is *CONTAINER*
Return values:	On success: backup level On failure: an empty string

Method:	**firstrun**
Syntax:	CalendarEvent <name> firstrun [value]
Description:	Set/get the absolute time in seconds (Posix time) for the first backup run. All subsequent runs will be scheduled starting from this time and using the *frequency* and *exception* rules.
Return values:	On success: time in seconds On failure: an empty string

Method:	**duration**
Syntax:	CalendarEvent <name> duration [value]
Description:	Set/get the backup duration in hours
Return values:	On success: number of hours On failure: an empty string

Method:	**frequency**
Syntax:	CalendarEvent <name> frequency [value]

Description:	Set/get the backup frequency determining how often the backup should be run. The [value] is a time description string as explained above.
Return values:	On success: time description string On failure: an empty string

Method:	**exception**
Syntax:	CalendarEvent <name> exception [value]
Description:	Set/get the backup exception following the same rules as *Backup <name> frequency.* Exception determines when a backup scheduled by the *frequency* should be skipped instead.
Return values:	On success: time description string On failure: an empty string

File Filter

File filter commands allow configuration of independent file filtering rules that can be applied to a backup plan to precisely manage which files should be included in the operation.

General commands

Method:	**create**
Syntax:	Filter create
Description:	Creates a new file filter
Return values:	On success: new filter *name* On failure: an empty string

Method:	**delete**
Syntax:	Filter <name> delete
Description:	Deletes the filter
Return values:	On success: the string “1” On failure: an empty string

Status and Information

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

Method:	**describe**
Syntax:	Filter <name> describe [value]
Description:	Sets or gets a human-readable description for the <name> filter. The <name> is one of the elements returned by the *names* method. If the element does not have a description assigned, the command returns the 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:	**disabled**
Syntax:	Filter <name> disabled
Description:	Queries the `Disabled` status
Return values:	On success: the string "1" (the fitler is disabled) or "0" (not disabled) On failure: an empty string

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

Control Commands

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

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

Method:	**include**
Syntax:	Filter <name> include [expression]
Description:	Get or set the filter expression used to select the files matching the expression. Expression is a string consisting of one or more of the following options: **-type** [path | directory | file] **-name** "expression" (supports * placeholders) **-mdate** ["<" | "=" | ">"] <value> (Modification date. <value> is either a date or "+X" for older than X days or "-X" for newer than X days) **-size** ["<" | ">"] <size in KB>
Example:	`"-name IMG-* -type file -mdate -7"`
Return values:	On success: client name On failure: an empty string

Method:	**exclude**
Syntax:	Filter <name> exclude [expression]
Description:	Get or set the filter expression used to exclude the files matching the expression. Uses the same format as the 'include' command.
Example:	`"-size >2048"`
Return values:	On success: list of paths On failure: an empty string

Method:	**extendedinclude**
Syntax:	Filter <name> extendedinclude [expression]
Description:	Get or set the filter expression used to select the files matching the expression but using the options and expression syntax from the Unix 'find' command.
Example:	`"-name \"*.txt\" -o -name \"*.pdf\" -o -name \"*.doc\""`
Return values:	On success: string "1" or "0" On failure: an empty string

Method:	**extendedexclude**
Syntax:	Filter <name> extendedexclude [expression]
Description:	Get or set the filter expression used to exclude the files matching the expression but using the options and expression syntax from the Unix 'find' command.
Exclude:	`"-type d –empty"`
Return values:	On success: script path On failure: an empty string

Archiving and Restoring

ArchiveEntry

The archive entry represents one archived file. It is an opaque handle which P5 uses to quickly locate the file on the archive media and it's 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 details upon creation.

Method:	**handle**
Syntax:	ArchiveEntry handle <client> <path> [<database>]
Description:	Returns the properly formatted archive entry handle which can be used for restoring files archived over the P5 web GUI. The <client> is the name of the P5 client where the <path> resides. 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}`. Furthermore, if the <path> contains `{` and/or `}` chars themselves, you must escape them with a backslash `\` character. The optional <database> declares the name of the database where the file has been indexed. If omitted, the standard `Default-Archive` database is used. If no such database could be found in the current P5 configuration, an error is triggered.
Return values:	On success: the handle of the entry On failure: an empty string

Method:	**btime**
Syntax:	ArchiveEntry <handle> btime
Description:	Returns the list of backup/archive times in seconds (Posix time) for each instance of the given archive entry.
Return values:	On success: the list of backup times On failure: an empty string

Method:	**mtime**
Syntax:	ArchiveEntry <handle> mtime
Description:	Returns the list of modification times in seconds (Posix time) for each instance of the given archive entry.
Return values:	On success: the list of modification times On failure: an empty string

Method:	**meta / getmeta**
Syntax:	ArchiveEntry <handle> meta [<key>] ArchiveEntry <handle> getmeta [<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 <key> argument: the value of the given meta key without <key> argument: the list of all the meta keys and their values On failure: an empty string

Method:	**setmeta**
Syntax:	ArchiveEntry <handle> setmeta [<key> <value> [<key> <value>]..]
Description:	Sets the defined meta-data key/value pair for the given archive entry. Key argument 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: the newly set key/value pair On failure: an empty string

Method:	**size**
Syntax:	ArchiveEntry <handle> size
Description:	Returns the list of sizes in bytes for each instance of the given archive entry.
Return values:	On success: the list of file sizes On failure: an empty string

Method:	**status**
Syntax:	ArchiveEntry <handle> status
Description:	Returns the status of the archived entry. An archive entry can have number of internal statuses, depending on the stage of the archive and/or restore process. Currently, the following statuses are supported: • indexed found in the archive index • unknown not found in the archive index The `indexed` status means that the entry has been processed (archived) and its meta data may be obtained from the index database. The `unknown` status means that the entry has not (yet) been found in the index, which is normal for files still waiting to be archived. If the status of an entry returns `unknown`, then all of the subsequent entry methods described below will return invalid values.
Return values:	On success: one of the supported statuses On failure: an empty string

Method:	**volume**
Syntax:	ArchiveEntry <handle> volume
Description:	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.
Return values:	On success: the ID of the volume if the entry was stored on only one volume, or a list of volume ID's if the entry was stored on multiple volumes On failure: an empty string

Preview/clip related

Method:	**clippath**
Syntax:	ArchiveEntry <handle> clippath [newpath]
Description:	If *newpath* is not given, the command will return the path of an existing clip or the string "*unknown"* if there is no clip available. If *newpath* is given as empty string "", it will clean/delete the previous clip (if any) and return the string "*unknown"* as a result. If *newpath* is given as a path to an existing file, this file will be set as the entry's clip. The file itself will be moved (not copied!) into the clip storage of the corresponding index and the absolute path of of the clip will be returned.
Return values:	On success: the path to the existing clip or the string "*unknown"* if not found On failure: an empty string

Method:	**clipurl**
Syntax:	ArchiveEntry <handle> clipurl <host> <port>
Description:	Returns a URL of the clip of the file as `http://host:client/url-to-the-clip` `<host>` and `<port>` refer to the host address and port of the P5 server host.
Return values:	On success: the URL as a string 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 P5 system, so the caller needs not (nor it should) perform any other task with the same resource.

Usage:

To use the ArchiveSelection resource, use the *create* method to create a new instance. After 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 submission, the resource goes out of scope and should not be used any more.

Method:	**create**
Syntax:	ArchiveSelection create <client> <plan> [<indexroot>]
Description:	Creates a new temporary archive selection resource. The resource will be automatically deleted after the associated archive job has been submitted. The <client> must be the one of the registered client computers on the current P5 server. You can get the list of client computers with the `Client names` CLI command. All files added with the *addentry* method (below) must reside on this client. The <plan> must be one of the registered archive plans. You can get the list of archive plans with the `ArchivePlan names` CLI command. The optional <indexroot> argument, if given, will force all files in the archive selection to be indexed under the <indexroot> path.
Return values:	On success: the name of the new resource. Use this name to address this resource in all other methods. On failure: an empty string

Method:	**addfrom**
Syntax:	ArchiveSelection <name> addfrom <input file> <output file>
Description:	Loads the Archive Selection entries from the external file <input file>. The file must be formatted with one entry per line, each entry in the format of: *<path>TAB<key1>TAB<value1>TAB<key2>TAB<value2>...* The <path> needs to be resolvable on the client for which the selection is created and the <input file> needs to reside on that client. The <path> may be followed by zero or more key/value pairs representing metadata that will be assigned to the file. All keys must be known in the index referenced by the archive selection. Unknown keys will be silently skipped. The <output file> is created by this command, it contains all accepted files with their ArchiveEntry handles used to reference the files later. The file format is one file per line in the format of: *<path>TAB<handle>* Note that unlike *ArchiveSelection addentry,* this method will add folders as empty nodes. This means: • folders are added without content, metadata in that case is assigned only to the folder • If files are added into a non existing folder in the archive, the folder is created without attributes or metadata.
Return values:	On success: the number of added key/value pairs On failure: an empty string

Method:	**addentry**
Syntax:	ArchiveSelection <name> addentry <path> [<key> <value> [<key> <value>]..]
Description:	Adds a single 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). The path will be stripped of the leading directory part and the name will be inserted into the index at the `indexroot` destination as defined in *create*. 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. To each path, you can assign an arbitrary number of <key> and <value> pairs. Those are saved in the archive index and can be used for searches during restore (see `RestoreSelection`). 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. In case the ArchiveSelection is set to incremental level and the given entry is already part of the Archive, the entry is not added and a string <empty> is returned.
Return values:	On success: the name of the new ArchiveEntry resource. This name must be used with `ArchiveEntry` methods to get the status and other meta-information for the entry after the archive operation has been completed. Please see the `ArchiveEntry` resource description On failure: an empty string

Method:	**addentryabs**
Syntax:	ArchiveSelection <name> addentryabs <path> [<key> <value> [<key> <value>]..]
Description:	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). The entry path will be added 1:1 into the index. Any prefixes and alternative index destinations are ignored. 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. To each path, you can assign an arbitrary number of <key> and <value> pairs. Those are saved in the archive index and can be used for searches during restore (see `RestoreSelection`). 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.
Return values:	On success: 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 On failure: an empty string

Method:	**adddirectory**
Syntax:	ArchiveSelection <name> adddirectory <path> [<key> <value> [<key> <value>]..]
Description:	Adds a 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). The path will be stripped of the leading directory part and the name will be inserted into the index at the `indexroot` destination as defined in *create*. 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` description for return values On failure: an empty string

Method:	**adddirectoryabs**
Syntax:	ArchiveSelection <name> adddirectoryabs <path> [<key> <value> [<key> <value>]..]
Description:	Adds a 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). The directory path will be added 1:1 into the index. Any prefixes and alternative index destinations are ignored. 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 for return values On failure: an empty string

Method:	**addfile**
Syntax:	ArchiveSelection <name> addfile <path> [<key> <value> [<key> <value>]..]
Description:	Adds a new file <path> to the archive selection <name>. It expects the absolute path to the file to be archived. The file must be located on the client <client> given at the resource creation time (see the `create` method). The path will be stripped of the leading directory part and the name will be inserted into the index at the `indexroot` destination as defined in *create*. See the `addentry method` description for explanation of other method arguments.
Return values:	On success: see the `addentry` method for return values On failure: an empty string

Method:	**addfileabs**
Syntax:	ArchiveSelection <name> addfileabs <path> [<key> <value> [<key> <value>]..]
Description:	Adds a new file <path> to the archive selection <name>. It expects the absolute path to the file to be archived. The file must be located on the client <client> given at the resource creation time (see the `create` method). The directory path will be added 1:1 into the index. Any prefixes and alternative index destinations are ignored. See the `addentry method` description for explanation of other method arguments.
Return values:	On success: see the `addentry` method for return values On failure: an empty string

Method:	**describe**
Syntax:	ArchiveSelection <name> describe [title]
Description:	If a title is given, the title is set as the description in the job monitor. The method returns the current description
Return values:	On success: the descriptions string as used in the job monitor 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: the string "0" (destroyed) the string "1" (not destroyed) On failure: an empty string

Method:	**entries**
Syntax:	ArchiveSelection <name> entries
Description:	Returns the number of entries in the selection object.
Return values:	On success: the number of entries On failure: an empty string

Method:	**level**
Syntax:	ArchiveSelection <name> [level]
Description:	Returns the level of the ArchiveSelection. If the optional level value is given, that level is set. The level must be either “*full*” or “*increment*”.
Return values:	On success: the string “*full*” or “*increment*” On failure: an empty string

Method:	**size**
Syntax:	ArchiveSelection <name> size
Description:	Returns the number of entries in the selection object. **This method is deprecated, please use ArchiveSelection entries instead.**
Return values:	On success: the number of entries On failure: an empty string

Method:	**submit**
Syntax:	ArchiveSelection <name> submit [<now>]
Description:	Submits the archive selection for execution. You can optionally override plan execution times by giving the <now> as one of the strings "1", "*t*", "*true*", "*True*", "*y*", "*yes*", or "*Yes*". 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.
Return values:	On success: 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. On failure: an empty string

Method:	**onjobactivation**
Syntax:	ArchiveSelection <name> onjobactivation <command>]

Description:	Registers the <command> to be executed just before the job is started by the *submit* method. The command itself can be any valid OS command plus variable number of arguments.
	The very first argument of the command (the program itself) can be prepended with the name of the P5 client where the command is to be executed on.
	If omitted, the command will be executed on the client which the ArchiveSelection object is created for.
	
	Examples:
	`ArchiveSelection 10002 onjobactivation "mickey:/var/scripts/myscript arg"`
	will execute /var/scripts/myscript on the client "mickey" regardless of the client the ArchiveSelection is created for. The program will be passed one argument: arg.
	`ArchiveSelection 10002 onjobactivation "/var/scripts/myscript"`
	will execute /var/scripts/myscript on the client the ArchiveSelection is created for.
	`ArchiveSelection 10002 onjobactivation "localhost:/var/scripts/myscript"`
	will execute /var/scripts/myscript on the P5 server.

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

Method:	**onjobcompletion**
Syntax:	ArchiveSelection <name> onjobcompletion <command>
Description:	Registers the <command> to be executed immediately after the job created by the *submit* method is completed. See *onjobactivation* for further information.
Return values:	On success: the command string On failure: an empty string

Method:	**onfiledeletion**
Syntax:	ArchiveSelection <name> onfiledeletion <command>
Description:	Registers the <command> to be executed immediately after the files are deleted through a job created by the *submit* method. See *onjobactivation* for further information.
Return values:	On success: the command string On failure: an empty string

ArchiveIndex

Queries P5 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 meta-data 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 databases and you can create new ones. If you need full control of ArchiveIndex resources, please use the P5 Web GUI.

General

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

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

Method:	**backup**
Syntax:	ArchiveIndex <name> backup <filename>
Description:	Produces the backup of the <name> archive index and saves the backup file as <filename>.
Return values:	On success: the file name 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 the backup of the database (see the *backup* methog).
Return values:	On success: the name of the backup file On failure: an empty string

Meta data Access

Method:	**addkey**
Syntax:	ArchiveIndex <name> addkey <key> <type> [<attr> <value>...]
Description:	Adds a user-defined key in the given index. The <key> identifier must not contain blanks, special punctuation characters nor any national characters. The length of the <key> identifier must 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 defined 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 P5 in any way, except for being stored in the key definition in the archive index.
Return values:	On success: the names of all the configured keys On failure: an empty string

Method:	**delkey**
Syntax:	ArchiveIndex <name> delkey <key>
Description:	Deletes a user-defined key in the given index.
Return values:	On success: the names of all the deleted keys On failure: an empty string

Method:	**keys**
Syntax:	ArchiveIndex <name> keys
Description:	Reports all the user-defined meta keys for the index <name>. Meta keys are used to store user-given meta-data to selected elements of the archive index.
Return values:	On success: a list of keys the string "<*empty*>" if no keys were defined On failure: an empty string

Method:	**keyget**
Syntax:	ArchiveIndex <name> keyget <key> [<attr>]
Description:	Returns the attributes for the given <key>. If no optional <attr> is supplied, all the defined attributes and their values as a list of key/value pairs is returned. If the <attr> is supplied, the value of the <attr> attribute is returned. Each <key> has at least the *type* one attribute. Please see the *addkey* method for description of the *type* attribute.
Return values:	On success: either a list of all the 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:	**keyhas**
Syntax:	ArchiveIndex <name> keyhas <key> <attr>
Description:	Checks whether the <key> has attribute <attr> defined
Return values:	On success: the string "1" if yes, or "0" otherwise 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 *type* attribute cannot be set. Please see the *addkey* method for a description of the *type* attribute.
Return values:	On success: the string "1" if the <attr> was set to the given value <val> or the string "0" if the key could not be set or if it does not exist On failure: an empty string

Method:	**inventory**
Syntax:	ArchiveIndex <name> inventory <output file> [<options>]
Description:	Outputs a list of the files contained in the Archive Index <name> into a file. The <output file> must be in the form [`client:`]`absolute_path` whereby `client` is the name of the P5 client where to store the file and `absolute_path` is the complete path to the file to hold the output. The client part is optional and defaults to `localhost:` The *inventory* command fills in the passed file with lines containing records separated by a TAB. If no <options> are given, the output file will by default contain the index paths of all the files saved by the given job <name>, one record per line. Additional <options> represent the attributes that will be output for each file in a tab-separated format. These attributes may be system attributes or any user-defined meta-data fields. The supported system attributes are: ppath: the physical path of the file on the filesystem volumes: a blank separated list of the volumes where the file is saved size: the size of the saved file handle: the handle as required by the RestoreSelection btime: the backup time of the file mtime: the file's modification time ino: the inode number of the file

The index path returned by the *inventory* command cannot be used to access files on the file system in general. There are special cases where this might be used for this purpose, but generally it is not supported. The idea behind this info is to have an overview or idea what is being stored in the index and not to consume it in some other fashion (i.e. address the files on the file system to post-process them).

In cases where files are still expected to be in the file system at the place they were at the point of archiving (for example somebody wants to delete them or otherwise post-process them) the ppath attribute may be used, which, when given on the command line, will yield the physical path as-found on the client where the file resides. Note that not all index entries have corresponding physical paths. In such cases the value will be set to empty.

Return values:

On success: the <client>:<output file>
On failure: an empty string

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 P5 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 a 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 P5 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> file system. All files will be placed in this directory instead of their original location. The <relocate> directory must exist on the client.
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> [<path>]
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`. By providing the optional path argument, it is possible to specify the target path of the restored file.
Return values:	On success: the path to the file to be restored Note: the returned path is not translated to match the optional <relocate> argument given at resource creation. On failure: an empty string

Method:	**addfrom**
Syntax:	RestoreSelection <name> addfrom <input file>
Description:	Loads the Restore Selection entries from the external file <input file>. The file must be formatted with one entry per line, each entry in the format of: <archiveentry>[TAB<relocate path>] The <archiveentry> is a handle to the archived file as returned by `ArchiveSelection addentry`. In case a <relocate path> is given, the archived file or folder is restored at the given path. Otherwise the relocate path as given in the RestoreSelection is used.
Return values:	On success: The count of entries that will be restored On failure: an empty string

Method:	**addfromvolume**
Syntax:	RestoreSelection <name> addfromvolume <volume>
Description:	Loads the Restore Selection entries from the given <volume>. The <volume> must be a volume ID known in P5, see the `Volume names command`. P5 will collect all files from the given volume, even if these are stored in different indexes. It is possible to add further files to the RestoreSelection with additional calls to `addentry, addfrom` or `addfromvolume` in order to for instance restore files from multiple volumes with a single command / job.
Return values:	On success: The found volume ID On failure: an empty string

Method:	**describe**
Syntax:	RestoreSelection <name> describe [title]
Description:	If a title is given, the title is set as description in the job monitor. The method returns the current description
Return values:	On success: the description string as used in the job monitor 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: the string "0" (destroyed) or "1" (not destroyed) On failure: an empty string

Method:	**entries**
Syntax:	RestoreSelection <name> entries
Description:	Returns the number of entries belonging to the restore selection <name>.
Return values:	On success: the number of entries On failure: an empty string

Method:	**findentry**
Syntax:	RestoreSelection <name> findentry <plan> {<expr>}
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 **Examples**: {author *= marco && state == italy} To search for files or folders by filename, the key *name* must be used. {name == myfile.pdf} or {name *= 'my file'} On Windows hosts, the expression must be additionally enclosed in quotation marks: “{name == myfile.pdf}” When entering expressions, please put curly braces around the complete expression. Values in expressions can be enclosed in single quotes, in case the value contains one or more blanks, it must be enclosed in single quotes. 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 *inaccessible* volume (the 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: the number of entries in the selection On failure: an empty string

Method:	**onfilecreation**
Syntax:	RestoreSelection <name> onfilecreation <command>
Description:	Registers the <command> to be executed immediately after the files are created through a job created by the *submit* method. See *onobjectactivation* for further information.
Return values:	On success: the command string On failure: an empty string

Method:	**onjobcompletion**
Syntax:	RestoreSelection <name> onjobcompletion <command>
Description:	Registers the <command> to be executed immediately after the job created by the *submit* method is completed. See *onobjectactivation* for further information.
Return values:	On success: the command string On failure: an empty string

Method:	**onjobactivation**
Syntax:	RestoreSelection <name> onjobactivation <command>]

Description:	Registers the <command> to be executed just before the job is started by the [submit] method. The command itself can be any valid OS command plus variable number of arguments.
	The very first argument of the command (the program itself) can be prepended with the name of the P5 client where the command is to be executed on.
	If omitted, the command will be executed on the client which the `RestoreSelection` object is created for.
	
	Examples:
	`RestoreSelection RestoreSelection.0 onjobactivation "mickey:/var/scripts/myscript arg"`
	will execute /var/scripts/myscript on the client "mickey" regardless what client the RestoreSelection is created for. The program will be passed one argument: arg.
	`RestoreSelection RestoreSelection.0 onjobactivation "/var/scripts/myscript"`
	will execute /var/scripts/myscript on the client the `RestoreSelection` is created for.
	`RestoreSelection RestoreSelection.0 onjobactivation "localhost:/var/scripts/myscript"`
	will execute /var/scripts/myscript on the P5 server.

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

Method:	**size**
Syntax:	RestoreSelection <name> size
Description:	Returns the summed up size in bytes of all files to restore.
Return values:	On success: the size in bytes 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 date in seconds since Jan 01, 1970 (Posix time).
Return values:	On success: the restore job ID. Use this job ID to query the status of the job by using Job resource. See the `Job resource` description for details. On failure: an empty string

Method:	**volumes**
Syntax:	RestoreSelection <name> volumes
Description:	Returns the media volume ID where the entries belonging to the restore selection <name> have 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.
Return values:	On success: a list of volume ID's containing all the entries On failure: an empty string

Media and Device related Commands

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 a list of single tape device resources.
Return values:	On success: the list of device names the string "<*empty*>" if no devices are configured On failure: an empty string

Method:	**cleaning**
Syntax:	Device <name> cleaning [value]
Description:	Sets or returns the value of the device cleaning flag. If the optional argument *value* is specified, it will be used to set the value of the flag. The argument must be 1 or 0 to set the cleaning flag on or off. If the optional argument is not specified it will return the current value of the flag.
Return values:	On success: the string "1" or "0" On failure: an empty string

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

Jukebox

This resource tracks jukeboxes configured for data storage. Currently you do not have much control over jukeboxes, except for getting the list of currently loaded volumes, resetting the jukebox and performing a bar code 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 a list of names of all jukebox resources
Return values:	On success: list of jukebox names the string "<*empty*>" If no jukeboxes are configured On failure: an empty string

Method:	**inventory**
Syntax:	Jukebox <name> inventory [-barcode [<startSlot> [<endSlot>]]]
Description:	Performs an inventory of the jukebox <name>, effectively updating the internal volume database. If the optional -barcode argument is specified, it attempts a bar code 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

Method:	**label**
Syntax:	Jukebox <name> label <pool> <slotID1> [<slotID2> … [<slotIDx>]]
Description:	Labels media in the given jukebox for the given POOL starting with slotID1, optionally including all of the slotIDs given on the command line. Example: Jukebox changer0 label My-Archive 1 5 9 this command will label the volumes in slots 1, 5 and 9 for pool MyArchive. Note that only new/empty volumes can be labeled with this command. Use the Job .. commands to monitor the ongoing label job.
Return values:	On success: the job Id of the label job On failure: an empty string

Method:	**slotcount**
Syntax:	Jukebox <name> slotcount
Description:	Returns number of media slots in the given jukebox. The slots in the Jukebox are addressed as 1 … slotcount.
Return values:	On success: the number of media slots On failure: an empty string

Method:	**reset**
Syntax:	Jukebox <name> reset
Description:	Performs a hardware jukebox reset, with forcefully emptying all jukebox drives. Use this method with caution since this command will perform an unconditional jukebox reset regardless of any jobs that may be using the jukebox resources.
Return values:	On success: the string "1" On failure: an empty string

Method:	**volumes**
Syntax:	Jukebox <name> volumes [<slotID>]
Description:	Returns a list of all volumes by id currently loaded in the <name> jukebox. In case a slotID is given, the command returns the volume in that slot. Note that slot IDs are numbered starting from 1, the id may differ from the numbering scheme of the library's web interface. To update the list of the volumes in the jukebox, use the *inventory* method. The volume names returned can be used as input for the Volume commands. In case a volume is present but unknown, a 0 is returned for that volume.
Return values:	On success: the list of volume names 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 P5 server. The preparation of media includes writing of the special label on the beginning of media. By using this label, the P5 server can uniquely identify the media in its volume database.

Method:	**names**
Syntax:	Volume names
Description:	Returns a list of names of all volume resources
Return values:	On success: the list of volume names the string "<*empty*>" if no volumes were configured On failure: an empty string

Method:	**barcode**
Syntax:	Volume <name> barcode
Description:	Returns the barcode of the volume <name>.
Return values:	On success: the barcode the string "<*empty*>" if no barcode is present On failure: the an empty string

Method:	**copyof**
Syntax:	Volume <name> copyof
Description:	Returns the volume name of the clone of this volume
Return values:	On success: the clone name or 0 (zero) if no clone exists On failure: an empty string

Method:	**dateexpires**
Syntax:	Volume <name> dateexpires
Description:	Returns the date when the volume will exxpire and can be relabeled in seconds since Jan 01, 1970 (Posix time).
Return values:	On success: the date in seconds (Posix time) On failure: an empty string

Method:	**dateused**
Syntax:	Volume <name> dateused
Description:	Returns the date when the volume was last used (for reading or for writing) in seconds since Jan 01, 1970 (Posix time).
Return values:	On success: the date in seconds (Posix time) On failure: an empty string

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

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

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

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

Method:	**isonline**
Syntax:	Volume <name> isonline
Description:	Returns the string "1" if the volume is accessible, being either in the media changer or in one of the media drives.
Return values:	On success: the string "1" On failure: an empty string

Method:	**inventory**
Syntax:	Volume <name> inventory <output file> [<options>]
Description:	Outputs a list of the files contained on the Archive-Volume <name> into a file. The <output file> must be in the form [`client:`]`absolute_path` whereby `client` is the name of the P5 client where to store the file and `absolute_path` is the complete path to the file to hold the output. The client part is optional and defaults to `localhost:` The *inventory* command fills in the passed file with lines containing records separated by a TAB. If no <options> are given, the output file will by default contain the index paths of all the files saved by the given job <name>, one record per line. Additional <options> represent the attributes that will be output for each file in a tab-separated format. These attributes may be system attributes or any user-defined meta-data fields. Note: This command can only be applied to Archive tapes The supported system attributes are: ppath: the physical path of the file on the filesystem size: the size of the saved file handle: the handle as required by the RestoreSelection btime: the backup time of the file mtime: the file's modification time ino: the inode number of the file The index path returned by the *inventory* command cannot be used to access files on the file system in general. There are special cases where this might be used for this purpose, but generally it is not supported. The idea behind this info is to have an overview or idea what is being stored in the index and not to consume it in some other fashion (i.e. address the files on the file system to post-process them). In cases where files are still expected to be in the file system at the place they were at the point of archiving (for example somebody wants to delete them or otherwise post-process them) the *ppath* attribute may be used, which, when given on the command line, will yield the physical path as-found on the client where the file resides. Note that not all index entries have corresponding physical paths. In such cases the value will be set to the string "<*empty*>".
Return values:	On success: the <client>:<output file> On failure: an empty string

Method:	**jobs**
Syntax:	Volume <name> jobs
Description:	Returns a list of job ids which accessed volume <name> The job ids can be used in a job command to get info about that job.
Return values:	On success: the job list On failure: an empty string

Method:	**label**
Syntax:	Volume <name> label [<value>]
Description:	Returns a human-readable description of the volume <name>. If the optional argument <value> is given, it will set the label to the given value. If optional argument <value> contains spaces it should be inside {} braces
Return values:	On success: the volume label On failure: an empty string

Method:	**location**
Syntax:	Volume <name> location [<value>]
Description:	Returns the physical location of the volume <name>. If the optional argument <value> is given, it will set the offline location parameter to the given value. If optional argument <value> contains spaces it should be inside {} braces. The format of the location is passed as name-of-the-jukebox : slot
Return values:	On success: the location string the string "<*empty*>" if the volume location is not set 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
Return values:	On success: the media type 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 *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:	**mode**
Syntax:	Volume <name> mode [<value>]
Description:	Returns the current mode of the volume <name>. The mode can be one of: • Appendable • Closed • Readonly • Recyclable • Full If the optional argument <value> is given, it will set the mode to the given value.
Return values:	On success: the volume mode On failure: an empty string

Method:	**state**
Syntax:	Volume <name> state [<value>]
Description:	Returns the current state of the volume <name>. The state can be one of: • Ok • Suspect • OutOfSync If the optional argument <value> is given, it will set the state to the given value.
Return values:	On success: the volume state On failure: an empty string

Method:	**totalsize**
Syntax:	Volume <name> totalsize
Description:	Returns the estimated capacity for the volume <name> in kbytes. The true capacity is variable and depends on the wear and tear and the number of faulty blocks on the volume and degrades with time and usage.
Return values:	On success: the number of kbytes On failure: an empty string

Method:	**usage**
Syntax:	Volume <name> usage
Description:	Returns the current usage of the volume <name>. Currently, the following usage types 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:	**usecount**
Syntax:	Volume <name> usecount
Description:	Returns the number of uses for read and/or write operations.
Return values:	On success: the number of uses On failure: an empty string

Method:	**usetime**
Syntax:	Volume <name> usedtime
Description:	Returns the total time that the volume has been used
Return values:	On success: the number of seconds 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:	**hardWrErCnt**
Syntax:	Volume <name> hardWrErCnt
Description:	Returns the number of nonrecovered write errors for the volume.
Return values:	On success: the number of errors On failure: an empty string

Method:	**softWrErCnt**
Syntax:	Volume <name> softWrErCnt
Description:	Returns the number of recovered write errors for the volume.
Return values:	On success: the number of errors On failure: an empty string

Method:	**hardRdErCnt**
Syntax:	Volume <name> hardRdErCnt
Description:	Returns the number of nonrecovered read errors for the volume.
Return values:	On success: the number of errors On failure: an empty string

Method:	**softRdErCnt**
Syntax:	Volume <name> softRdErCnt
Description:	Returns the number of recovered read errors for the volume.
Return values:	On success: the number of errors On failure: an empty string

Pool

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

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

Method:	**create**
Syntax:	Pool create <name> [<option> <value>]
Description:	Creates a media pool with the name <name>. The <name> of the pool may not include blanks or any special punctuation and/or national characters. If the pool <name> already exists in the P5 configuration an error will be thrown. Options supported by this command are: usage one of *Archive* or *Backup* mediatype one of *TAPE* or *DISK* *blocksize* *count* If no optional arguments are given, the newly created pool will be assigned *Archive* for usage and *TAPE* for media type. The new option blocksize <count> allows to specify blocksize for all volumes labeled for this pool. The <count> parameter can be as low as 32768 (32K) and as high as 524288 (512K) but it must be one of: 32768, 65536, 131072, 262144, 524288 The newly created pool will be configured for no parallelism i.e. it will use only one media-device for writing and/or reading the media. If you need to configure the pool for parallelism, use method Pool drivecount. Example to create tape-archive media pool: `Pool create MyPool usage Archive mediatype TAPE`
Return values:	On success: the name of the created pool On failure: an empty string

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

Method:	**drivecount**
Syntax:	Pool <name> drivecount <count>
Description:	Sets the drives per stream the pool is allowed to use
Return values:	On success: the "1" (the pool is disabled) or "0" (not disabled) On failure: an empty string

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

Method:	**mediatype**
Syntax:	Pool <name> mediatype
Description:	returns one of *TAPE* or *DISK* designating the media type of labeled volumes in the pool.
Return values:	On success: the media-type as a string On failure: an empty string

Method:	**totalsize**
Syntax:	Pool <name> totalsize
Description:	Returns the estimated capacity for the pool <name> in kbytes. The true capacity is variable and depends on the wear and tear and the number of faulty blocks on the volume and degrades with time and usage.
Return values:	On success: the number of kbytes On failure: an empty string

Method:	**usage**
Syntax:	Pool <name> usage
Description:	Returns either *Archive* or *Backup*
Return values:	On success: the usage as a string On failure: an empty string

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

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

Job related Commands

Job

The Job resource tracks jobs submitted to the P5 server. Information about each of the submitted jobs is held indefinitely and can be queried by the user at any time. Job resources are generated automatically, for instance by the *submit* methods of the ArchiveSelection resource.

General

Method:	**names**	
Syntax:	Job names	
Description:	Returns a list of all currently scheduled or running jobs	
Return values:	On success:	the names of currently scheduled or running jobs the string "<*empty*>" if no jobs are scheduled
	On failure:	an empty string

Status and Information

Method:	**completed**	
Syntax:	Job completed [<lastdays>]	
Description:	Returns the names of all jobs completed by the system. If the optional <lastdays> argument is not given, jobs completed today are returned. Otherwise, all completed jobs for the last <lastdays> days are returned. The <lastdays> argument is interpreted as a positive integer (the default is 0 meaning today).	
Return values:	On success:	the names of completed jobs or the string "<*empty*>" if no jobs completed in the given time.
	On failure:	an empty string

Method:	**completion**
Syntax:	Job <name> completion
Description:	Returns the completion code of the completed job. The completion code can be one of: • success • warning • exception • failure The `success` completion code means that the job has completed successfully in its entirety. It means that all of the files have been archived and/or restored, though. For info about the particular file, use the *protocol* method. The `warning` completion code means that the job came to a regular end, but it is incomplete. At least one file could not be saved. For info about the particular file, use the *protocol* method. The `exception` completion code means that parts of the job have failed, but the job may have been partially executed successfully. This happens for parallel archive/restore operations where one of the job threads runs into an error, while others continue to run and finish successfully. The `failure` completion code means that the job has failed in its entirety and none of the files have been processed (archived/restored) correctly.
Return values:	On success: one of the completion codes On failure: an empty string

Method:	**describe**
Syntax:	Job <name> describe
Description:	Returns a (human readable) job description as shown in the P5 job monitor.
Return values:	On success: the job description On failure: an empty string

Method:	**failed**
Syntax:	Job failed [<lastdays>]
Description:	Returns the names of all the jobs that failed to execute. If no optional argument <lastdays> is given, it returns jobs that failed today. Otherwise, all failed jobs for the last <lastdays> days are returned. The <lastdays> argument is interpreted as a positive integer (0 means today).
Return values:	On success: the names of failed jobs the string "<*empty*>" if no jobs failed On failure: an empty string

Method:	**inventory**
Syntax:	Job <name> inventory <output file> [<options>]
Description:	Outputs a list of the files saved by the Archive-Job <name> into a file. The <output file> must be in the form [`client:`]`absolute_path` whereby `client` is the name of the P5 client where to store the file and `absolute_path` is the complete path to the file to hold the output. The client part is optional and defaults to `localhost:` The *inventory* command fills in the passed file with lines containing records separated by a TAB. If no <options> are given, the output file will by default contain the index paths of all the files saved by the given job <name>, one record per line. Additional <options> represent the attributes that will be output for each file in a tab-separated format. These attributes may be system attributes or any user-defined meta-data fields. The supported system attributes are: ppath: the physical path of the file on the filesystem volumes: a blank separated list of the volumes where the file is saved size: the size of the saved file handle: the handle as required by the RestoreSelection btime: the backup time of the file mtime: the file's modification time ino: the inode number of the file The index path returned by the *inventory* command cannot be used to access files on the file system in general. There are special cases where this might be used for this purpose, but generally it is not supported. The idea behind this info is to have an overview or idea what is being stored in the index and not to consume it in some other fashion (i.e. address the files

	on the file system to post-process them). In cases where files are still expected to be in the file system at the place they were at the point of archiving (for example somebody wants to delete them or otherwise post-process them) the *ppath* attribute may be used, which, when given on the command line, will yield the physical path as-found on the client where the file resides. Note that not all index entries have corresponding physical paths. In such cases the value will be set to the string "<*empty*>*"*.
Return values:	On success: the <client>:<output file> On failure: an empty string

Method:	**label**
Syntax:	Job <name> label
Description:	Returns the (human readable) job label. The following labels are returned: *Archive*, *Backup*, *Synchronize* and *System.* A `Job label` can be used in conjunction with the `Job describe` command to better display the job record in various list displays.
Return values:	On success: the job label On failure: an empty string

Method:	**pending**
Syntax:	Job pending
Description:	Returns the names of all the jobs waiting to be executed, i.e. jobs that are still in the queue waiting to be scheduled and jobs that are already scheduled but wait for the next free worker thread.
Return values:	On success: the names of currently waiting jobs the string "<*empty*>" if no jobs are waiting On failure: an empty string

Method:	**protocol**
Syntax:	Job <name> protocol [<archiveentry>]
Description:	Returns a 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:	**report**
Syntax:	Job <name> report
Description:	Returns a 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:	**resourcegroup**
Syntax:	Job <name> resourcegroup
Description:	Returns the name of the resource group for which this job has been running.
Return values:	On success: the name of the resource group (for example `ArchivePlan`, `SyncPlan`, etc.) or the string "<*empty*>", if no resource group is 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: the name of the resource (for example `Default-Backup`, `Default-Archive`) or the string "<*empty*>", if no resource group is associated with the job On failure: an empty string

Method:	**running**
Syntax:	Job running
Description:	Returns the names of all currently running jobs.
Return values:	On success: the names of currently running jobs the string "<*empty*>" if no jobs are running On failure: an empty string

Method:	**status**
Syntax:	Job <name> status
Description:	Returns the status of the job. A job can have a number of internal statuses, depending on the stage of the archive and/or restore process. Currently, the following statuses are supported: • started the job is starting (intermediate state) • stopped the job is stopping (intermediate state) • unknown the job is not known by the system • scheduled the job is in the queue waiting to be run • pending an intermediate state during start, the job is waiting to be accepted for start by the queue manager • running the job is running • canceled the job is canceled by user • completed the job is completed • terminated the job is terminated by a server shutdown
Return values:	On success: one of the supported statuses On failure: an empty string

Method:	**totalfiles**
Syntax:	Job <name> totalfiles
Description:	Returns the number of files and folders saved by the given archive of backup job
Return values:	On success: the number of files and folders saved On failure: an empty string

Method:	**totalkbytes**
Syntax:	Job <name> totalkbytes
Description:	Returns the amount of data saved by the given archive or backup job in kbyte
Return values:	On success: the total size of files and folders saved On failure: an empty string

Method:	**warning**
Syntax:	Job warning [<lastdays>]
Description:	Returns names of all jobs with warnings. If no optional argument <lastdays> is given, it returns jobs with warnings from today. Otherwise, all jobs with warnings for the last <lastdays> days are returned. The <lastdays> argument is interpreted as a positive integer (0 = today).
Return values:	On success: the names of jobs with warnings the string "<*empty*>" if no jobs ended with a warning On failure: an empty string

Method:	**xmlticket**
Syntax:	Job <name> xmlticket [<outfilename>]
Description:	Returns the 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

Method:	**cancel**
Syntax:	Job <name> cancel
Description:	Cancels the running job. Only jobs that have the `running` status can be canceled. An attempt to cancel a job with a different status will result in an error.
Return values:	On success: the string "1" if the job is canceled the string "0" if the job could not be canceled for whatever reason On failure: an empty string

Method:	**runat**
Syntax:	Job <name> runat
Description:	Returns the time in seconds (Posix time) when the job was scheduled to run.
Return values:	On success: the time On failure: an empty string

Method:	**stop**
Syntax:	Job <name> stop
Description:	Stops the scheduled job. Only jobs that have the `scheduled` status can be stopped. An attempt to stop a job with a different status will result in an error.
Return values:	On success: the string "1" if the job is stopped the string "0" if the job could not be stopped for whatever reason On failure: an empty string

Overview Commands

These commands return a JSON object containing the most often accessed information about the corresponding resource, similar to what one would get from the P5 GUI.

Method:	**backup2go**
Syntax:	Overview backup2go
Description:	Returns an overview of the current backup2go state of the configured workstations
Return values:	On success: JSON object On failure: an empty string

Example:	<pre>{
	  "Backup2GoOverview": [
	  {
	  "name": "10021",
	  "description": "awdbserver.local",
	  "start time": "2022-02-15T13:43:32Z",
	  "finish time": "2022-02-15T13:44:46Z",
	  "status": "finished",
	  "sizeKbytes": 663517,
	  "summary": "Okay (647.97 MB)",
	  "last successful": "2022-02-15T13:44:46Z",
	  "template": "Generic Template"
	  }
	  ]
	}</pre>

Method:	**backup**
Syntax:	Overview backup
Description:	Returns an overview of the current backup state and pool usage
Return values:	On success: JSON object On failure: an empty string
Example:	{


```
  "Backup Overview": [
    {
      "Client": "localhost",
      "Backup Plan": "10002",
      "Last Run": {
        "start time": "2022-02-24T19:35:00Z",
        "finish time": "2022-02-23T20:14:20Z",
        "status": "scheduled",
        "sizeKbytes": 3540542,
        "summary": "Finished (3.38 GB)"
      },
      "Last Successful": {
        "finish time": "2022-02-23T20:14:20Z",
        "status": "finished",
        "sizeKbytes": 3540542,
        "summary": "18 hours ago (3.38 GB)"
      },
      "Directories": [
        "/vol1/develop",
        "/vol2/backup_staging_area/blog",
        "/vol2/backup_staging_area/portal"
      ],
      "Last Pool": "PoolTwo",
      "Next Run": "2022-02-24T19:35:00Z",
      "Next Pool": [
        "PoolOne",
        "PoolTwo",
```

```
            "PoolThree"
          ]
        }
      ],
      "Pool Usage": [
        {
          "type": "container",
          "name": "Container volume Local_disk_arc",
          "usedBytes": 16749,
          "totalBytes": -1,
          "freeBytes": -1,
          "usedPercent": 0,
          "summary": "16.36 MB used / P5 Licensed: no size limit"
        }
      ]
    }
```

Method:	**archive**
Syntax:	Overview archive
Description:	Returns an overview of the current archive state of the configured workstations
Return values:	On success: JSON object On failure: an empty string
Example:	


```
{
  "Archive Overview": [
    {
      "plan": "10001",
      "start time": "2021-12-13T17:56:01Z",
      "finish time": "2021-12-13T17:57:05Z",
      "status": "finished",
      "sizeKbytes": 3958,
      "client": "localhost",
      "directories": [
        "/usr/local/aw/logs"
      ],
      "pool": "Local_disk_arc"
    }
  ],
  "Pool Usage": [
    {
      "type": "container",
      "name": "Container volume Local_disk_arc",
      "usedBytes": 16749,
      "totalBytes": -1,
      "freeBytes": -1,
      "usedPercent": 0,
      "summary": "16.36 MB used / P5 Licensed: no size limit"
    }
  ]
}
```

Method:	**synchronize**
Syntax:	Overview synchronize
Description:	Returns an overview of the current Synchronize state
Return values:	On success: JSON object On failure: an empty string
Example:	{


```
  "Synchronize Overview": [
    {
      "Syncronize Plan": "10007",
      "Description": "portal2stage",
      "Last Run": {
        "start time": "2022-02-28T05:17:01Z",
        "finish time": "2022-02-28T05:18:33Z",
        "status": "scheduled",
        "sizeKbytes": 0,
        "summary": "Error"
      },
      "Last Successful": {
        "start time": "2022-02-10T05:17:00Z",
        "finish time": "2022-02-10T05:20:01Z",
        "sizeKbytes": 216233,
        "summary": "18 days ago  (211.17 MB)"
      },
      "Next Run": "2022-03-01T00:17:00Z",
      "Source Host": "portal",
      "Source Path": [
        "/etc",
        "/var/www",
        "/usr/lib",
        "/home"
      ],
      "Target Host": "localhost",
      "Target Path": "/vol2/backup_staging_area/portal"
```

```
      }
    ]
  }
```

Examples

Interactive CLI usage

The following examples are made using the `nsdchat` utility from a shell script on the P5 server machine. The `nsdchat` utility is invoked in 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
```

Example: Volume List

The following script creates a csv formatted list of all volumes known in P5.
Reroute the output to a file named volumes.csv to create a file that can be opened with a spread sheet like Microsoft Excel or LibreOffice Calc.

```
#!/bin/sh
# Create a volume list
#
# Change the path in case P5 is installed elsewhere
chatcmd="/usr/local/aw/bin/nsdchat -c"

list=`$chatcmd Volume names`
echo "Label,Barcode,State,Mode,Type,'Used Size','Last Used',Location"

for i in $list
do
   c1=`$chatcmd Volume $i label`
   c2=`$chatcmd Volume $i barcode`
   c3=`$chatcmd Volume $i state`
   c4=`$chatcmd Volume $i mode`
   c5=`$chatcmd Volume $i mediatype`
   c6=`$chatcmd Volume $i usedsize`
   c7=`$chatcmd Volume $i dateused`
   c8=`$chatcmd Volume $i location`
   echo "'$c1','$c2','$c3','$c4','$c5','$c6','$c7','$c8'"
done
# EOF
```

Please note that this is a shell script that cannot be used on Windows.

Example: Workstation List

The following script creates a csv formatted list of the workstations and lists start and end time as well as file count and size of the last job and displays the totals.
Reroute the output to a file named workstations.csv to create a file that can be opened with a spread sheet like Microsoft Excel or LibreOffice Calc.

```
#!/bin/sh
#
# Change the path in case P5 is installed elsewhere
chatcmd="/usr/local/aw/bin/nsdchat -c"

list=`$chatcmd Workstation names`
echo "Name,Start,End,Files,Size"

for i in $list
do
   c1=`$chatcmd Workstation $i describe`
   c2=`$chatcmd Workstation $i lastbegin`
   c3=`$chatcmd Workstation $i lastend`
   c4=`$chatcmd Workstation $i totalfiles`
   c5=`$chatcmd Workstation $i totalkbytes`
   echo "'$c1','$c2','$c3','$c4','$c5'"
done

# EOF
```

Please note that this is a shell script that cannot be used on Windows.

The date values are given in seconds since 01.01.1970, 00:00. This is a different time base from the one used in spreadsheets so conversion must be done in case the date should be shown. For example by adding the shell command

```
c2=`date -r $c2'
```

before the last `echo` line in the above script, the date will be displayed in a readable format of a the seconds value.
Calculating lastend – lastbegin or ($c3 - $c2) renders the number of seconds the job took to complete. In case that value is negative, the last job did not succeed.

Example: Job List

The following script prints a list of failed backup job of the last 3 days.

```
#!/bin/sh
# List failed jobs of last days
# pass a "-w" to get jobs that gave warnings instead
#
# Change the path in case P5 is installed elsewhere
# Note that here the -s argument is used to pass connection
# parameters in the nsdchat call.

chatcmd="/usr/local/aw/bin/nsdchat -s awsock:/user:passwd@localhost:9001 -c"

if [ "$1" == "-w" ]
then
   list=`$chatcmd Job warning 3`
else
   list=`$chatcmd Job failed 3`
fi

count=0

for i in $list
do
   rg=`$chatcmd Job $i resourcegroup`
   if [ "$rg" != "::BackupTask" ]
   then
      continue
   fi
   let ++count
   echo Job     : `$chatcmd Job $i label`
   echo Status  : `$chatcmd Job $i status` with `$chatcmd Job $i completion`
   echo Protocol:
   echo `$chatcmd Job $i protocol`
   echo ------------------------------------------------------------
done
echo $count jobs in the last 3 days
# EOF
```

Please note that this is a shell script that cannot be used on Windows.
The script only outputs the backup jobs' descriptions (if any).

Example: Posix Time and Conversions

Several methods, mainly in the *Backup2Go* section of this manual, use Posix time to represent a time and date.

Posix time is native on Unix systems. It is the number of seconds since Jan 01, 1970.

The conversion between that format and a human readable format can easily be done in a Unix terminal session with the Unix date command:

From a readable format to Posix time:

```
% date -j -f "%Y-%m-%d %H:%M:%S" "2012-10-02 12:00:00" +%s
1349172000
```

From Posix time to a readable format:

```
% date -r 1349172000
Tue  2 Oct 2012 12:00:00 CEST
```

The highest date representable this way is Jan 19, 2038.
Higher values will be interpreted as being in the past:

```
% date -r 2147483647
Tue Jan 19 04:14:07 CET 2038
% date -r 2147483648
Fri Dec 13 20:45:52 WET 1901
```