OPSWARE INC
Automating IT™

Opsware® SAS Command Line Interface 2.0 Reference Guide

Corporate Headquarters
599 North Mathilda Avenue
Sunnyvale, California 94085 U.S.A.
T + 1 408.744.7300 F +1 408.744.7383 www.opsware.com

Copyright © 2000-2005 Opsware Inc. All Rights Reserved.

Opsware Inc. Unpublished Confidential Information. NOT for Redistribution. All Rights Reserved.

Opsware is protected by U.S. Patent Nos. 6,658,426, 6,751,702, 6,816,897, 6,763,361 and patents pending

Opsware, Opsware Command Center, Model Repository, Data Access Engine, Web Services Data Access Engine, Software Repository, Command Engine, Opsware Agent, Model Repository Multimaster Component, and Code Deployment & Rollback are trademarks and service marks of Opsware Inc. All other marks mentioned in this document are the property of their respective owners.

Additional proprietary information about third party and open source materials can be found at http://www.opsware.com/support/opensourcedoc.pdf.

Table of Contents

Preface v

Contents of this Guide v
Conventions in this Guide. vi
Icons in this Guide vi
Chapter 1: Installing and Configuring OCLI 1
Installing OCLI 1
Configuring OCLI 2
Password 3
Preparing OCLI 3
Shell Function for Unix 3
Path for Windows 4
Testing OCLI 4
Chapter 2: OCLI Commands 5
OCLI Command Line 5
Before Using the OCLI 5
Elements of OCLI Commands 7
OCLI Object Specifiers 7
OCLI Object Parameters. 7
OCLI Commands 7
Command Reference 9

assign Command 9
attach Command 10
concatenate Command. 11
copy Command. 12
detach Command. 14
disjunction Command 14
download Command 15
execute Command. 16
get Command 17
grant Command 18
help Command 19
intersection Command 20
list Command 20
login Command. 21
move Command 22
new Command 23
preview Command 24
provision Command. 25
read Command 25
reconcile Command. 27
revoke Command. 27
set Command 28
this Command. 29
unassign Command 29
union Command 30
upload Command. 31
write Command. 32

Object List 34

- APPLICATION Object 34
- CUSTOMER Object 34
- FACILITY Object 35
- JOB Object 35
- PACKAGE Object 36
- PLATFORM Object 36
- ROLE Object 37
- SERVER Object 37
- SERVICELEVEL Object 38
- TEMPLATE Object 38
- USER Object 39

OCLI Command Line 41

OCLI Command Syntax 42

- Simple Commands 42
- Chaining Commands 44
- Working with Custom Attributes 46
- Set Operations 46

Preface

Welcome to the *Opsware® SAS Command Line Interface 2.0 Reference Guide*. The OCLI is a command line interface to the Opsware Server Automation System (SAS). Using the information in this Reference Guide, you will be able to perform available Opsware SAS operations from the command line.

This product and this documentation is intended for use by senior system administrators, and assumes that the user has experience with command line interfaces, and that they know how to write an advanced batch file or shell script.

This version of the OCLI uses the term “object” to refer to familiar items that make up the Opsware model, such as User, Server, Application, and so on. This document contains a complete discussion of these objects and how to use them.

Contents of this Guide

This guide contains the following chapters and appendices:

Chapter 1: Installing and Configuring OCLI - provides information about the installation and configuration of OCLI, including what files to download and where to find them, how to set the environment variables, how to obtain needed certificates, and how to test the installation to make sure everything was installed correctly.

Chapter 2: The Command Line - provides detailed information about command line commands and their associated arguments, as well as an object list and the properties for each object.

Appendix A: OCLI Command Line Syntax Tutorial - provides information about OCLI commands and how to structure them, including examples of commands of varying degrees of complexity.

Conventions in this Guide

This guide uses the following typographical and formatting conventions.

NOTATION	DESCRIPTION
`Courier`	Identifies text of displayed messages and other output from Opsware programs or tools.
`Courier Bold`	Identifies user-entered text (commands or information).
`Courier Italics`	Identifies variable user-entered text on the command line or within example files.

Icons in this Guide

This guide uses the following iconographic conventions.

ICON	DESCRIPTION
📄	This icon is a note. It identifies especially important concepts that warrant added emphasis.
☑	This icon is a requirement. It identifies a task that must be performed before an action under discussion can be performed.
💡	This icon is a tip. It identifies information that can help simplify or clarify tasks.
⚠	This icon is a warning. It is used to identify significant information that must be read before proceeding.

Chapter 1: Installing and Configuring OCLI

IN THIS CHAPTER

This chapter describes:

- Installing OCLI
- Configuring OCLI
- Preparing OCLI
- Testing OCLI

Installing OCLI

There are two versions of OCLI. One version is to be installed on managed servers, and one version is to be installed on a Windows desktop.

1. Select the version of OCLI that you want to use. Select either the Windows desktop version, or the version to be installed on managed servers.

   If the servers have an agent installed, select the managed server version. This includes servers which host an Opsware core component.

   If you will be working on Windows systems that have no Opsware Agent installed, use the Windows 32 bit version.

   If the server you want to install it on has no agent and is not a Windows server, ask your Opsware, Inc. representative for a solution.

2. Download the OCLI packages from the Opsware, Inc. download site. This site requires a login and password, which you can get from your Opsware administrator.

3. Unzip your selected version and install it in a directory of your choice.

Configuring OCLI

A certificate is required to run OCLI. If you are running OCLI on a managed server or core, and you are executing OCLI as a user with sufficient local file system privileges, there is a certificate already available for use.

If you are running the stand alone version of OCLI, you will need to obtain a certificate. Ask your Opsware Administrator for the token.srv.

Once you have obtained the token.srv and are ready to work with the OCLI, please keep in mind that using this certificate with the OCLI invokes a different security mechanism than the one used by the Opsware Command Center. As a result, you may have increased or reduced privileges and you may be able to perform operations on servers belonging to customers that you may not normally have access to, and you may be able to perform operations that you may not normally be able to perform from within the Opsware Command Center. Consequently, use caution with the OCLI to avoid any unintended consequences that could arise with this increased level of permissions.

1. To connect to the Opsware core from a Windows system that is not managed by Opsware SAS, the OCLI needs the client certificate and key that were generated during the installation of Opsware SAS. The name of the certificate is token.srv and it is inside the opsware-cert.db generated during install. Ask your Opsware Administrator for this certificate and place it in the directory where you have OCLI installed.

2. Provide the URL of the core components that the OCLI uses to communicate with Opsware SAS. If you are running the OCLI on a managed server or core, these are determined automatically via DNS resolution of the names `spin`, `theword`, and `way.` If you are running the stand alone version of the OCLI, set the environment variables listed in the following table.

Table 1-1: OCLI Environment Variables

NAME	COMPONENT	DEFAULT VALUE
`OPSWARE_SPINURL`	Data Access Engine	`https://spin:1004`
`OPSWARE_WAYURL`	Command Engine	`https://way:1018`
`OPSWARE_WORDURL`	Software Repository	`https://theword:1003`

For example, if you have a lab core with the IP address 192.0.2.1, you would need to set `OPSWARE_SPINURL` to the value `https://192.0.2.1:1004`, `OPSWARE_WAYURL` to the value `https://192.0.2.1:1018`, and `OPSWARE_WORDURL` to the value `https://192.0.2.1:1003`.

To set these values on Windows, use the commands:

```
set OPSWARE_SPINURL=”https://<hostname>:1004”
set OPSWARE_WAYURL=”https://<hostname>:1018”
set OPSWARE_WORDURL=”https://<hostname>:1003”
```

(where `<hostname>` is the hostname or IP address of the system that is running the appropriate core component).

To set these values on a Unix system while running a Bourne shell (sh), use the commands:

```
OPSWARE_SPINURL=”https://<hostname>:1004”; export SPINURL
OPSWARE_WAYURL=”https://<hostname>:1018”; export WAYURL
OPSWARE_WORDURL=”https://<hostname>:1003”; export WORDURL
```

(modify these commands as necessary depending on the shell you are using)

Password

If you are using the OCLI in a script or batch file, and you wish to have that script automatically authenticate with a username and password, you may set the password via an environment variable. If the environment variable `OPSWARE_PASSWORD` is set, OCLI will use that when executing the login command.

Preparing OCLI

Shell Function for Unix

If you are a Unix user, the OCLI function must be defined. This is a shell function that will be called when you invoke OCLI from your shell. If you are using bourne shell (or the equivalent), enter the following command:

```
eval `./ocli_setup.py`
```

Path for Windows

If you are running a Windows operating system, it is recommended that `ocli.cmd` be in your system search path. If your OCLI is installed in `c:\ocli`, you could add this directory to your search path with the command:

```
set PATH=%PATH%;c:\ocli
```

Testing OCLI

If you want to test that your OCLI has been installed correctly, change to the directory where the OCLI was installed, and enter the following command:

```
ocli
```

You should see the introductory help page to indicate that the OCLI and all of its components have been installed correctly.

If you want to test that your OCLI is communicating properly with the Opsware core, enter the following command:

```
ocli CUSTOMER any
```

If the communication with the core is working properly, the OCLI will return `--customer=0`

Chapter 2: OCLI Commands

IN THIS CHAPTER

This chapter includes the following topics:

- OCLI Command Line
- Before Using the OCLI
- Elements of OCLI Commands
- Command Reference
- Object List

OCLI Command Line

The OCLI command line utility is intended as an alternative to using the Opsware Command Center. You can perform virtually every Opsware SAS function from the command line with this utility, with the exception of creating users and customers. These two functions must be performed from within the Opsware Command Center.

Before Using the OCLI

The OCLI is a command line interface to Opsware SAS. In order to use it, you must first:

- Download and install the OCLI client as described in Chapter 1, Installing and Configuring the OCLI.
- If you are running on a Windows system that is not managed by Opsware SAS, make sure that the necessary certificate for communicating with the Opsware core is in the directory where the OCLI was installed.
- For convenience, if you are a Windows user, you may optionally place `ocli.cmd` in the executable search path.

- Make sure that you can communicate with the Opsware core by ensuring that the Opsware DNS names—`spin`, `way`, and `theword`—can be resolved, or by setting the appropriate environment variables. (See “Configuring OCLI” on page 2 of this guide.)
- Make sure, if you are a Unix user, to define the OCLI function by executing the command:

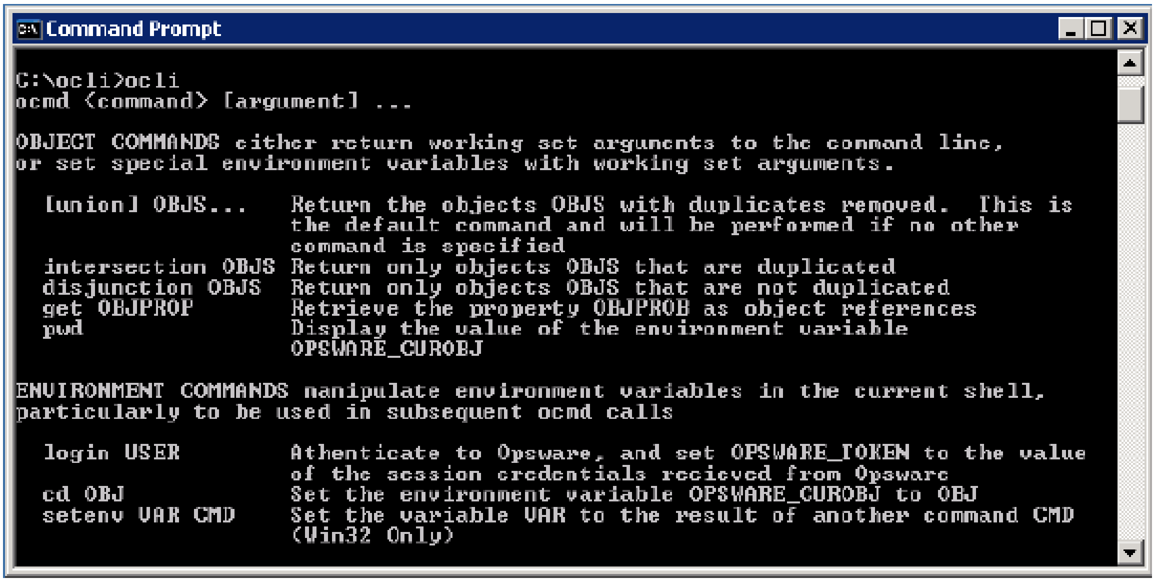
  ```
  eval `./ocli_setup.py`
  ```

  This needs to be done every time you use OCLI in a new shell process.
- Test that the OCLI and all of its components are installed correctly by executing the following command:

  ```
  % ocli
  ```

  if everything is installed correctly, the system will return several screens of information about using OCLI, as shown in Figure 2-1, below.

Figure 2-1: Results of Testing Successful OCLI Installation



```
C:\ocli>ocli
ocmd <command> [argument] ...

OBJECT COMMANDS either return working set arguments to the command line,
or set special environment variables with working set arguments.

  [union] OBJS...    Return the objects OBJS with duplicates removed.  This is
                     the default command and will be performed if no other
                     command is specified
  intersection OBJS  Return only objects OBJS that are duplicated
  disjunction OBJS   Return only objects OBJS that are not duplicated
  get OBJPROP        Retrieve the property OBJPROB as object references
  pwd                Display the value of the environment variable
                     OPSWARE_CUROBJ

ENVIRONMENT COMMANDS manipulate environment variables in the current shell,
particularly to be used in subsequent ocmd calls

  login USER         Athenticate to Opsware, and set OPSWARE_TOKEN to the value
                     of the session credentials recieved from Opsware
  cd OBJ             Set the environment variable OPSWARE_CUROBJ to OBJ
  setenv VAR CMD     Set the variable VAR to the result of another command CMD
                     (Win32 Only)
```

- Test that your OCLI is communicating properly with the Opsware core by executing the following command, which is used to discover the customer number of the customer called “any.”:

  ```
  % ocli CUSTOMER any
  ```

  If communication with the core is working properly, OCLI will return the message `--customer=0`

Elements of OCLI Commands

OCLI Object Specifiers

The “object specifiers” in this syntax are familiar to Opsware SAS users:

- APPLICATION
- CUSTOMER
- FACILITY
- JOB
- PACKAGE
- PLATFORM
- ROLE
- SERVER
- SERVICELEVEL
- TEMPLATE
- USER

OCLI Object Parameters

OCLI object parameters are the representation that is used by OCLI to identify Opsware objects. Each object has a unique combination of object type and object ID that serves as a primary key. Many Opsware commands return object parameters as output which can be piped to other commands to perform compound actions. They may also be assigned to environment variables, copied to a file, or cut and pasted into additional commands, as desired. These IDs are unique to each installation of Opsware SAS, and are not portable between cores that do not share a Model Repository.

OCLI Commands

The commands that can be used with the OCLI are:

- assign
- attach
- concatenate
- copy

- detach
- disjunction
- download
- execute
- get
- grant
- help
- intersection
- list
- login
- move
- new
- preview
- provision
- read
- reconcile
- revoke
- set
- this
- unassign
- union
- upload
- write

This chapter contains a complete description of each command, plus examples. For a tutorial about syntax and usage, refer to Appendix A, OCLI Command Syntax.

Command Reference

Examples are provided for each of the commands available for use with the OCLI. The syntax and usage of most commands are the same for both Unix and Windows platforms. The examples provided are for Unix; in those cases where the syntax and usage differs, an example is provided for both Unix and Windows. Also, due to the formatting of this document, some long commands in some examples may wrap to a second line. When entering the actual command, make sure that the command is entered all on one line.

assign Command

The `assign` command is used to assign packages to application nodes. Every package passed into this command will be paired and compared with every application node, and if they are compatible, the package will be assigned to the application node.

Usage

```
    ocli <PACKAGE>... <APPLICATION>... assign
```

Arguments

Object parameters may be provided at the stdin.

PACKAGE is an object parameter representing the packages to be attached. See `ocli help PACKAGE` for more information. There may be one or more of these parameters.

APPLICATION is an object parameter representing the application nodes that the packages will be attached to. See `ocli help APPLICATION` for more information. There may be one or more of these parameters.

Security and Limitations

You cannot assign packages to application nodes if the application node is of OS version "OS Independent". Also, you cannot assign a package that has an OS version of "OS Independent" or is of type "Unknown".

The package OS version and the application node OS version must be identical. If they are not identical, they will silently be skipped.

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to manipulate the model. All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to assign a package with ID 324099 to an application node with ID 9650100.

```
# ocli --package=324099 --application=9650100 assign
#
```

In this example, we want the OCLI to find all packages that start with “foo” and assign them to application nodes “OS Extras / bar” and “Other Applications / baz”. Note that package assignments will only be made if the Customer and Platform attributes are compatible.

```
# ocli PACKAGE foo* | ocli APPLICATION "OS Extras / bar" "Other
Applications / baz" | ocli assign
# ocli APPLICATION "OS Extras / bar" | ocli get PACKAGES | ocli
get name
foo-1.2.3.rpm
foo-4.5.6.pkg
# ocli APPLICATION "Other Applications / baz" | ocli get
PACKAGES | ocli get name
foo-7.8.9.zip
foo_for_windows.msi
```

attach Command

The attach command is used to attach servers to application nodes. Every server passed into this command will be paired and compared with every application node, and if they are compatible, the server will be attached to the application node.

Usage

```
ocli <PACKAGE>... <APPLICATION>... attach
```

Arguments

Object parameters may be provided at the stdin.

SERVER is an object parameter representing a managed server. There may be one or more of these parameters.

APPLICATION is an object parameter representing the application nodes that will be installed on the server. See `ocli help APPLICATION` for more information. There may be one or more of these parameters.

Security and Limitations

You cannot assign applications to servers if the application node is of OS version “OS Independent”.

The server OS version and the application node OS version must be identical. If they are not identical, they will silently be skipped.

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to manipulate the model. All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to attach an application node with ID 9650100 to a server with ID 3328101.

```
# ocli --application=9650100 --server=3328101 attach
#
```

This example asks the OCLI to find all servers in the domain example.com and attach the application nodes “OS Extras / bar” and “Other Applications / baz”. Note that application attachments will only be made if the Customer and Platform attributes are compatible.

```
# ocli APPLICATION "OS Extras / bar" "Other Applications / baz"
| SERVER *.example.com | ocli attach
# ocli SERVER foo.example.com | ocli get APPLICATIONS | ocli get
pathname
OS Extras / bar
Other Applications / baz
...
...
#
```

concatenate Command

The concatenate command is the default, and will be executed if object specifiers are provided without a command explicitly given. The concatenate command appends any object parameters from the stdin, and object parameters from the command line, as well as object specifiers passed as arguments, and returns the results to the stdout as object parameters. If there are duplicate objects provided, the duplicates will be preserved by this operation. You may also use the alias `plus` for this command.

Usage

```
ocli [OBJS] concatenate [SPECIFIER]...
ocli [OBJS] plus [SPECIFIER]...
ocli [OBJS] <SPECIFIER> [SPECIFIER]... (the command is
implied)
```

Arguments

Object parameters may be provided at the stdin.

OBJS consists of one or more object parameters (such as --template=22400099, --server=95030099, --customer=0, etc.) with each parameter consisting of --, the object name, =, and the object ID. This argument is optional for this command.

A SPECIFIER consists of an object label (such as SERVER, APPLICATION, PACKAGE, etc.) followed by one or more search terms. Any number of SPECIFIER arguments may be provided to this command.

Security and Limitations

All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to concatenate the application with ID 9650100 with a server with ID 3328101.

```
# ocli --application=9650100 | ocli --server=3328101
--application=9650100 --server=3328101
#
```

In this example, we ask OCLI to concatenate the server foo.example.com, the application “OS Extras / bar” and the package baz-1.2.3.rpm. Note that, because concatenate is the default command, it does not need to be explicitly stated; in this example, the concatenate command is implied.

```
# ocli SERVER foo.example.com | ocli APPLICATION "OS Extras /
bar" | ocli PACKAGE baz-1.2.3.rpm
--server=3328101 --application=9650100 --package=324099
# ocli --server=3328101 --application=9650100 --package=324099 |
ocli get name
foo
bar
baz-1.2.3.rpm
#
```

copy Command

This command allows you to copy objects in the Model Repository. The types of objects that may be copied are Application, Service Level, and Template.

Usage

```
    ocli < APPLICATION | SERVICELEVEL | TEMPLATE > [--deep ] copy
    < DESTINATION >
```

Arguments

Object parameters may be provided at the stdin.

APPLICATION is an object parameter representing an application node to be copied. See `ocli help APPLICATION` for more information. There may be only one parameter and if an application parameter is provided, no other parameters may be supplied.

SERVICELEVEL is an object parameter representing a service level to be copied. See `ocli help SERVICELEVEL` for more information. There may be only one parameter and if a service level parameter is provided, no other parameters may be supplied.

TEMPLATE is an object parameter representing a template to be copied. See `ocli help TEMPLATE` for more information. There may be only one parameter and if a template parameter is provided, no other parameters may be supplied.

You may specify the --deep option to copy objects recursively.

A DESTINATION consists of the full pathname of the object destination. Use the same format for the name that you would use for a specifier. See help on the individual objects for details.

Security and Limitations

There can be at most one source object to copy from. You may specify a single destination to copy to.

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to manipulate the model.

Examples

This example shows how to copy the application with ID 9650100 to the new location, “Other Applications / Application Foo / Version 1.2.3".

```
# ocli --application=9650100 copy "Other Applications /
Application Foo / Version 1.2.3"
--application=5430099
#
```

This example asks OCLI to copy a service level called “Service Levels / foo / old” to one called “Service Levels / foo / new”.

```
# ocli SERVICELEVEL "Service Levels / foo / old" | ocli copy
"Service Levels / foo / new"
--servicelevel=4323099
# ocli --servicelevel=4323099 get name
new
# ocli --servicelevel=4323099 get fullname
Service Levels foo new
# ocli --servicelevel=4323099 get pathname
```

```
Service Levels / foo / new
#
```

detach Command

The detach command is used to detach packages from application nodes. Every package passed into this command will be paired and compared with every application node, and if they are currently attached, the package will be detached from the application node.

Usage

```
    ocli <PACKAGE>... <APPLICATION>... detach
```

Arguments

Object parameters may be provided at the stdin.

PACKAGE is an object parameter representing the packages to be detached. See `ocli help PACKAGE` for more information. There may be one or more of these parameters.

APPLICATION is an object parameter representing the application nodes that the packages will be detached from. See `ocli help APPLICATION` for more information. There may be one or more of these parameters.

Security and Limitations

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to manipulate the model.

Examples

This example shows how to detach an application node with ID 9650100 from a server with ID 3328101.

```
# ocli --application=9650100 --server=3328101 detach
#
```

In this example, OCLI is asked to detach the application “OS Extras / Application Foo / Version 1.5" from the server “foo.example.com”.

```
# ocli SERVER foo.example.com | ocli APPLICATION "OS Extras /
Application Foo / Version 1.5" | ocli detach
```

disjunction Command

The disjunction command takes the object parameters from the stdin, and object parameters from the command line, as well as object specifiers passed as arguments, and returns any objects that are not duplicated. You may also use the alias `xor` for this command.

Usage

```
ocli [OBJS] disjunction [SPECIFIER]...
ocli [OBJS] xor [SPECIFIER]...
```

Arguments

Object parameters may be provided at the stdin.

OBJS consists of one or more object parameters (such as --template=22400099, --server=95030099, --customer=0, etc.) with each parameter consisting of --, the object name, =, and the object ID. This argument is optional for this command.

A SPECIFIER consists of an object label (such as SERVER, APPLICATION, PACKAGE, etc.) followed by one or more search terms. Any number of SPECIFIER arguments may be provided to this command.

Security and Limitations

All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to find all non-duplicated objects in a list of objects.

```
# ocli --application=9650100 --server=3328101 --server=3328101
--customer=0 --customer=0 --template=22400099 disjunction
--application=9650100 --template=22400099
#
```

In this example, OCLI is asked to get a listing of all packages attached to application “OS Extras / foo” that are not attached to application “OS Extras / bar”.

```
# ocli APPLICATION "OS Extras / foo" "OS Extras / bar" "OS
Extras / bar" | ocli get PACKAGES | ocli xor
--package=324099 --package=863099
#
```

download Command

The download command is used to retrieve packages from the Software Repository.

Usage

```
ocli < PACKAGE >... download
```

Arguments

Object parameters may be provided at the stdin.

PACKAGE is an object parameter representing the packages to be downloaded. See `ocli help PACKAGE` for more information. There may be one or more of these parameters.

Security and Limitations

This command must be run on a server running an Opsware component, or a managed server that has been assigned to the customer Opsware, or you must log in as a user with the appropriate Command Engine (way) permissions.

Examples

This example shows how to download a package.

```
# ocli --package=863099 download
#
```

In this example, OCLI is asked to download each of the packages assigned to applications “OS Extras / foo” and “OS Extras / bar” using the `or` alias to indicate download only once.

```
# ocli APPLICATION "OS Extras / foo" "OS Extras / bar"| ocli get
PACKAGES | ocli or | ocli download

#
```

execute Command

The execute command is used run a remote script. The contents of the script is read from the stdin. The output of the script is returned to the stdout.

Usage

```
    ocli < SERVER > execute
```

Arguments

Object parameters may be provided at the stdin.

SERVER is an object parameter representing a managed server. There should be one or more of these parameters.

Security and Limitations

This command must be run on a server running an Opsware component, or you must log in as a user with the appropriate Command Engine (way) permissions.

Examples

This example shows how to perform a directory listing on a Unix server.

```
# export srv=`ocli SERVER unix.example.com`
# echo $srv
--server=3328101
# ocli $srv execute
ls /
^D
bin   cust  etc   initrd  lib        misc  mnt_isos  proc  sbin
usr
boot  dev   home  lc      lost+found  mnt   opt     root  tmp
var
#
```

In this example, OCLI is being asked to perform a directory listing on a Windows server.

```
C:\ocli>ocli SERVER windows.example.com
--server=3240104

C:\ocli>ocli --server=3240104 execute
dir c:\
^Z
 Volume in drive C has no label.
 Volume Serial Number is 10AC-8DF2

 Directory of c:\

08/23/2004  02:46 PM    <DIR>          ATI
09/03/2002  01:36 PM                 0 AUTOEXEC.BAT
09/03/2002  01:36 PM                 0 CONFIG.SYS
07/12/2004  01:53 PM    <DIR>          cygwin
11/23/2004  04:06 AM               164 Debug.log
12/09/2004  09:26 AM    <DIR>          Documents and Settings
06/02/2003  12:07 PM    <DIR>          I386
10/21/2004  02:13 PM    <DIR>          j2sdk1.4.2_06
12/24/2004  01:14 PM    <DIR>          Program Files
12/13/2004  01:32 PM    <DIR>          Python24
12/19/2004  08:59 PM    <DIR>          temp
12/30/2004  04:38 PM    <DIR>          WINDOWS
               3 File(s)            164 bytes
              19 Dir(s)   7,597,600,768 bytes free

C:\ocli>
```

get Command

The get command retrieves object properties from the Model Repository. The value of the property is written to the stdout.

Usage

```
ocli [ OBJS ] get [ PROPERTY ]...
```

Arguments

OBJS consists of one or more object parameters (such as --template=22400099, --server=95030099, --customer=0, etc.) with each parameter consisting of --, the object name, =, and the object ID. This argument is optional for this command.

PROPERTY consists of a space delimited list of property names to be retrieved. The contents of each property will be retrieved, and output as a space delimited list. The properties for each object will be printed on separate lines.

Security and Limitations

All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to get the name and ID properties from objects.

```
# ocli --application=9650100 --server=3328101 --customer=0
--template=2240099 get name id
foo 9650100
bar.example.com 3328101
Customer Independent 0
baz 2240099
#
```

In this example, OCLI is asked to get a listing of all packages attached to application “OS Extras / foo” that are not attached to application “OS Extras / bar”.

```
# ocli PLATFORM "SunOS 5.8" "SunOS 5.9" | ocli get SERVERS |
ocli get name hostname ip use stage
oberon oberon.example.com 10.128.34.2 UNKNOWN UNKNOWN
io io.example.com 10.128.34.36 DEVELOPMENT LIVE
#
```

grant Command

The grant command is used to grant Command Engine (way) privileges (in the form of roles) to users. Every role passed into this command will be paired and compared with every user, and the user will be granted the permissions of that role.

Usage

```
ocli < USER >... < ROLE >... grant
```

Arguments

Object parameters may be provided at the stdin.

USER is an object parameter representing a user known to the Command Engine (way). There should be one or more of these parameters. These arguments may be provided at the stdin.

ROLE is an object parameter representing a Command Engine (way) permission. There should be one or more of these parameters. These arguments may be provided at the stdin.

Security and Limitations

You must authenticate using the login command as a user with administrative privileges before using this command.

Examples

This example shows how to grant the permissions represented by the role with ID 20199 to the user with ID 10199.

```
# ocli --role=20199 --user=10199 grant
#
```

This example shows how to grant all of Joe's permissions to Mary.

```
# ocli USER joe | ocli get ROLES | ocli USER mary | ocli grant
#
```

help Command

The help command provides command line help for the OCLI.

Usage

```
    ocli help [ COMMAND | OBJECT ]
```

Arguments

If you do not provide an argument to the help command, you will be provided with a summary of OCLI help. If you provide the name of a command, you will be provided with detailed help for that command. If you provide an object type, you will be provided with detailed information about that object.

intersection Command

The intersection command takes the object parameters from the stdin, and object parameters from the command line, as well as object specifiers passed as arguments, and returns any objects that are duplicated at least once. You may also use the alias `and` for this command.

Usage

```
ocli [ OBJS ] intersection [ SPECIFIER ]...
ocli [ OBJS ] and [ SPECIFIER ]...
```

Arguments

OBJS consists of one or more object parameters (such as --template=22400099, --server=95030099, --customer=0, etc.) with each parameter consisting of --, the object name, =, and the object ID. This argument is optional for this command.

A SPECIFIER consists of an object label (such as SERVER, APPLICATION, PACKAGE, etc.) followed by one or more search terms. Any number of SPECIFIER arguments may be provided to this command.

Security and Limitations

All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to find all duplicated objects in a list of objects.

```
# ocli --application=9650100 --server=3328101 --server=3328101
--customer=0 --customer=0 --template=22400099 intersection
--server=3328101 --customer=0
#
```

In this example, OCLI is asked to get a listing of all packages attached to application “OS Extras / foo” that are also attached to application “OS Extras / bar”.

```
# ocli APPLICATION "OS Extras / foo" "OS Extras / bar" | ocli
get PACKAGES | ocli and
--package=5430100 --package=5420100
#
```

list Command

The list command returns a list of objects. It is not recommended that you frequently retrieve a list of objects with over a thousand members.

Usage

```
    ocli list [ OBJECTS ]
```

Arguments

OBJECTS is the plural name of the object you retrieve a list of. Valid options are SERVERS, PACKAGES, CUSTOMERS, PLATFORMS, FACILITIES, APPLICATIONS, SUBNODES, TEMPLATES, JOBS, USERS, ROLES, SERVICELEVELS, SERVERGROUPS, and OPERATINGSYSTEMS.

Security and Limitations

All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to list all customers.

```
# ocli list CUSTOMERS
--customer=0 --customer=9 --customer=15 --customer=100
#
```

This example shows how to list all platforms.

```
# ocli list PLATFORMS
SunOS 5.6
SunOS 5.7
SunOS 5.8
SunOS 5.9
AIX 4.3
 ...
SuSE Linux 8.0
#
```

login Command

The login command authenticates a user to Opsware SAS, and sets a cryptographic token that will serve as authentication credentials for further OCLI commands. This token, stored in the environment variable OPSWARE_TOKEN, is used to determine privilege in subsequent OCLI commands. To switch users, re-run the OCLI login command, or to log out (and return to the base set of privileges provided by your certificate), clear the value of OPSWARE_TOKEN from your shell.

Usage

```
    ocli login [ USERNAME ]
```

Arguments

USERNAME is the user with which you want to authenticate. The user's privileges are determined by the Command Engine (way), rather than by the rights granted in the Opsware Command Center. Privileges may be modified through the use of the grant and revoke commands.

Security and Limitations

You must have a valid username and Command Engine (way) permissions for the actions you are trying to perform for this command to be successful.

Examples

This example shows how to log in as user John.

```
# ocli login john
Enter Password: ******
#
```

move Command

This command allows you to move objects from one parent node to another node in the Model Repository. The types of objects that may be moved are Application, Service Level, and Template.

Usage

```
ocli < APPLICATION | SERVICELEVEL | TEMPLATE > move
<DESTINATION>
```

Arguments

Object parameters may be provided at the stdin.

APPLICATION is an object parameter representing an application node to be moved. See `ocli help APPLICATION` for more information. There may be only one parameter and if an application parameter is provided, no other parameters may be supplied.

SERVICELEVEL is an object parameter representing a service level to be moved. See `ocli help SERVICELEVEL` for more information. There may be only one parameter and if a service level parameter is provided, no other parameters may be supplied.

TEMPLATE is an object parameter representing a template to be moved. See `ocli help TEMPLATE` for more information. There may be only one parameter and if a template parameter is provided, no other parameters may be supplied.

A DESTINATION consists of the full pathname of the object destination. Use the same format for the name that you would use for a specifier. See help on the individual objects for details.

Security and Limitations

There can be at most one source object to move. You may specify a single destination to move the object to.

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to manipulate the model.

Examples

This example shows how to move the application with ID 9650100 to the new location, “Other Applications / Application Foo / Version 1.2.3".

```
# ocli --application=9650100 move "Other Applications /
Application Foo / Version 1.2.3"
--application=9650100
#
```

This example shows how to move a service level called “Service Levels / foo / old” to a service level called “Service Levels / foo / new”.

```
# ocli SERVICELEVEL "Service Levels / foo / old" | ocli move
"Service Levels / foo / new"
--servicelevel=4323099
# ocli --servicelevel=4323099 get name
new
# ocli --servicelevel=4323099 get fullname
Service Levels foo new
# ocli --servicelevel=4323099 get pathname
Service Levels / foo / new
#
```

new Command

The new command creates a new object. The only Object types that can be created are APPLICATION, TEMPLATE, and SERVICELEVEL.

Usage

```
   ocli new [ OBJECT ] NAME ...
```

Arguments

OBJECT is the name of the object you are creating. Valid objects are APPLICATION, TEMPLATE, and SERVICELEVEL.

NAME is the name of the new object you are creating. It must be expressed as a full path, and the parent node to this new object must already exist.

Security and Limitations

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to manipulate the model.

Examples

This example shows how to create a new application called “Other Applications / Application Foo / Version 1.2.3".

```
# ocli new APPLICATION "Other Applications / Application Foo /
Version 1.2.3"
--application=9650100
#
```

This example shows how to create a new service level called “Service Levels / foo / bar”.

```
# ocli new SERVICELEVEL "Service Levels / foo / bar"
--servicelevel=4323099
#
```

preview Command

The preview command allows you to preview a reconcile, and will compare the server's current state against the model to determine what packages should be installed or uninstalled.

Usage

```
    ocli <SERVER> preview
```

Arguments

These arguments may be provided at the stdin.

SERVER is an object parameter representing a managed server. There should be one of these parameters.

Security and Limitations

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to run the Reconcile wayscript.

Examples

This example shows how to perform a preview reconcile on a server with the ID of 5030099.

```
# ocli --server=5030099 reconcile
```

provision Command

The provision command allows you to install an operating system on a server in the server pool.

Usage

```
ocli <CUSTOMER> <TEMPLATE> <SERVER>... provision
```

Arguments

These arguments may be provided at the stdin.

CUSTOMER is an object parameter representing the customer that the server being provisioned should be assigned to, or customer ANY if a customer assignment is not desired. The customer parameter is required, and there must be one, and only one customer parameter.

TEMPLATE is an object parameter representing the template that will be used to provision the server. There must be one, and only one template parameter.

SERVER is an object parameter representing a server in the pool. There should be one or more of these parameters.

Security and Limitations

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to run an OS Provisioning wayscript.

Examples

This example shows how to provision an operating system on a server with the ID of 95030099, using the template with an ID of 92384099, and assigning the server to the customer with an ID of 0.

```
# ocli --customer=0 --template=92384099 --server=39480102
provision

#
```

read Command

The read command gets custom attributes. The content of the custom attribute is written to the stdout.

Usage

```
ocli [OBJS] read [ATTRIBUTE]...
```

Arguments

OBJS consists of one or more object parameters (such as --template=22400099, --server=95030099, --customer=0, etc.) with each parameter consisting of --, the object name, =, and the object ID. This argument is optional for this command. The only valid object parameters are of types that allow custom attributes. These are application, customer, facility, operating system, platform, server, server group, service level, and template.

ATTRIBUTE consists of a space-delimited list of custom attribute names to be set. The contents of each custom attribute will be retrieved, and out put as a space delimited list. The custom attributes for each object will be printed on separate lines.

Security and Limitations

All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to read the custom attribute “bar” from the application with ID 3094099.

```
# ocli --application=3094099 read bar
foo
#
```

This example asks the OCLI to read the custom attributes “state1”, “state2”, and “state3” from all child nodes of the service level “Service Levels / foo.”

```
# ocli SERVICELEVEL "Service Levels / foo"
--servicelevel=4323099
# ocli SERVICELEVEL "Service Levels / foo" | ocli get SUBNODES
--servicelevel=4543099 --servicelevel=4544099
--servicelevel=4545099
# ocli SERVICELEVEL "Service Levels / foo" | ocli get SUBNODES |
ocli get pathname
Service Levels / foo / group1
Service Levels / foo / group2
Service Levels / foo / group3
# ocli SERVICELEVEL "Service Levels / foo" | ocli get SUBNODES |
ocli read state1 state2 state3
installed ready running
not_installed not_ready cancelled
installed not_ready finished
```

reconcile Command

The reconcile command allows you to perform a reconcile, and install and uninstall packages on the managed server to bring it into compliance with the model.

Usage

```
ocli <SERVER> reconcile
```

Arguments

These arguments may be provided at the stdin.

SERVER is an object parameter representing a managed server. There should be one of these parameters.

Security and Limitations

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to run the Reconcile wayscript.

Examples

This example shows how to perform a reconcile on a server with the ID of 5030099.

```
# ocli --server=5030099 reconcile
```

revoke Command

The revoke command is used to revoke Command Engine (way) privileges (in the form of roles) to users. Every role passed into this command will be paired and compared with every user, and the user will have the permissions of that role revoked.

Usage

```
ocli < USER >... < ROLE >... revoke
```

Arguments

Object parameters may be provided at the stdin.

USER is an object parameter representing a user known to the Command Engine (way). There should be one or more of these parameters. These arguments may be provided at the stdin.

ROLE is an object parameter representing a Command Engine (way) permission. There should be one or more of these parameters. These arguments may be provided at the stdin.

Security and Limitations

You must authenticate by using the login command as a user with administrative privileges before using this command.

Examples

This example shows how to revoke the permissions represented by the role with ID 20199 from the user with ID 10199.

```
# ocli --role=20199 --user=10199 revoke
#
```

This example shows how to revoke way-supervisor privileges from the user named Bob.

```
# ocli USER bob ROLE way-supervisor | ocli revoke
#
```

set Command

The set command modifies object properties in the Model Repository.

Usage

```
    ocli [ OBJS ] get [ PROPERTY=VALUE ]...
```

Arguments

OBJS consists of one or more object parameters (such as --template=22400099, --server=95030099, --customer=0, etc.) with each parameter consisting of --, the object name, =, and the object ID. This argument is optional for this command.

PROPERTY consists of a property name. VALUE is the value you wish the property to be set to.

Security and Limitations

All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to set the name of the server represented by ID 20199 to “foo.example.com”.

```
ocli --server=20199 set name=foo.example.com
```

This example shows how to toggle the `is_folder` property of a template on and off.

```
# ocli TEMPLATE "Template / baz" | ocli get is_folder
0
# ocli TEMPLATE "Template / baz" | ocli set is_folder=1
# ocli TEMPLATE "Template / baz" | ocli get is_folder
```

```
1
# ocli TEMPLATE "Template / baz" | ocli set is_folder=
# ocli TEMPLATE "Template / baz" | ocli get is_folder
0
```

this Command

The this command returns the object parameters representing the current managed server or core that OCLI is executed on.

Usage

```
    ocli [OBJS] this
```

Arguments

Object parameters may be provided at the stdin.

OBJS consists of one or more object parameters. These are simply passed through and added to the server object parameter produced by this command.

Security and Limitations

This command does not produce usable output if it is run on a computer that is not managed by Opsware SAS.

Examples

This example shows how to get the server object representing the system that OCLI is currently executing on.

```
# ocli this
--server=20199
# ocli this | ocli get name
foo.example.com
```

unassign Command

The unassign command is used to remove packages from application nodes. Every package passed into this command will be paired and compared with every application node, and if that package is currently assigned to the application node, it will be removed.

Usage

```
    ocli <PACKAGE>... <APPLICATION>... unassign
```

Arguments

Object parameters may be provided at the stdin.

PACKAGE is an object parameter representing the packages to be attached. See `ocli help PACKAGE` for more information. There may be one or more of these parameters.

APPLICATION is an object parameter representing the application nodes that the packages will be attached to. See `ocli help APPLICATIO`N for more information. There may be one or more of these parameters.

Security and Limitations

You cannot assign packages to application nodes if the application node is of OS version "OS Independent". Also, you cannot assign a package that has an OS version of "OS Independent" or is of type "Unknown".

The package OS version and the application node OS version must be identical. If they are not identical, they will silently be skipped. All data lookup is limited by the permissions of your authentication credentials.

This command must be run on the core as root, or you must authenticate as a user with sufficient Command Engine (way) privileges to manipulate the model.

Examples

This example shows how to unassign a package with ID 324099 from an application node with ID 9650100.

```
# ocli --package=324099 --application=9650100 unassign
#
```

This example shows how to unassign all packages from “OS Extras / bar”.

```
# ocli APPLICATION "OS Extras / bar" | ocli get PACKAGES | ocli
APPLICATION "OS Extras / bar" | ocli unassign
#
```

union Command

The union command takes object parameters from the stdin, and object parameters from the command line, as well as object specifiers passed as arguments, and returns the set of results to the stdout as object parameters. The resulting set will not have any duplicate items. You may also use the alias `or` for this command.

Usage

```
   ocli [OBJS] union [SPECIFIER]...
   ocli [OBJS] or [SPECIFIER]...
```

Arguments

Object parameters may be provided at the stdin.

OBJS consists of one or more object parameters (such as --template=22400099, --server=95030099, --customer=0, etc.) with each parameter consisting of --, the object name, =, and the object ID. This argument is optional for this command.

A SPECIFIER consists of an object label (such as SERVER, APPLICATION, PACKAGE, etc.) followed by one or more search terms. Any number of SPECIFIER arguments may be provided to this command.

Security and Limitations

All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to return all objects in a list of objects, with no duplicates in the returned list.

```
# ocli --application=9650100 --server=3328101 --server=3328101
--customer=0 --customer=0 --template=22400099 union
--application=9650100 --server=3328101 --customer=0
--template=22400099
#
```

This example shows how to download each of the packages assigned to applications “OS Extras / foo” and “OS Extras / bar” using the `or` alias to indicate download only once.

```
# ocli APPLICATION "OS Extras / foo" "OS Extras / bar"| ocli get
PACKAGES | ocli or | ocli download

#
```

upload Command

The upload command is used to transfer packages into the Software Repository.

Usage

```
   ocli < CUSTOMER > < PLATFORM > [ --uploadfiletype=TYPE ]
   upload [ FILENAME ]...
```

Arguments

Object parameters may be provided at the stdin.

CUSTOMER is an object parameter representing the customer to which the newly uploaded package will be assigned. See `ocli help PACKAGE` for more information.

PLATFORM is an object parameter representing the platform that the uploaded package is for. See `ocli help PLATFORM` for more information.

TYPE is an optional file type extension, such as zip or rpm.

FILENAME consists of a space delimited list of filenames to be uploaded into Opsware SAS. They should be either full path names, or exist in the current directory. For every combination of customer and platform objects that are provided, OCLI will upload the package into the Software Repository using that combination of customer and platform.

Security and Limitations

This command must be run on a server running an Opsware component, or a managed server that has been assigned to the customer Opsware, or you must log in as a user with the appropriate Command Engine (way) permissions.

Examples

This example shows how to upload a package with filename of “foo.zip” using a customer with an ID of 0 and a platform with an ID of 120000.

```
# ocli --customer=0 --platform=120000 upload foo.zip

#
```

This example shows how to upload some packages for two customers on several platforms.

```
# ocli CUSTOMER any Opsware PLATFORM "Windows 2000" "Windows
2003" "Windows NT 4.0" | ocli upload foo.msi bar.zip

#
```

write Command

The write command sets custom attributes. The content of the custom attribute is read from the stdin.

Usage

```
ocli [ OBJS ] [ --datafile=FILE ] write [ ATTRIBUTE ]...
```

Arguments and Options

Object parameters may not be provided at the stdin, unless the datafile option is specified.

OBJS consists of one or more object parameters (such as --template=22400099, --server=95030099, --customer=0, etc.) with each parameter consisting of --, the object name, =, and the object ID. This argument is optional for this command. The only valid object parameters are of types that allow custom attributes. These are application, customer, facility, operating system, platform, server, server group, service level, and template.

ATTRIBUTE consists of a space delimited list of custom attribute names to be set. The content read from the stdin will be written to each attribute on each object.

The datafile option allows the data to be written to the custom attribute to come from a file rather than the stdin.

Security and Limitations

All data lookup is limited by the permissions of your authentication credentials.

Examples

This example shows how to write to the "Customer Independent" customer the value "bar" to the custom attribute "foo".

```
# echo bar | ocli --customer=0 write foo

#
```

This example shows how to write to the current system's custom attribute "cust" the name of the currently assigned customer.

```
# export this=`ocli this`

# echo $this
--server=20199

# ocli $this | ocli get CUSTOMER | ocli get name | ocli $this
write cust

# ocli $this read cust
Opsware
```

Object List

APPLICATION Object

The APPLICATION object represents a logical unit of software. When an application is installed on a server, all associated packages are installed, and any configuration changes associated with the application are also made.

Properties

- ID
- name
- fullname
- SUBNODES
- PACKAGES
- PLATFORM
- CUSTOMER
- SERVERS
- stackid
- requires_attachment
- notes
- allow_devices
- allow_multiple
- date_modified
- description
- locked

CUSTOMER Object

A Customer is an account within Opsware SAS that has access to designated resources, such as servers and software.

Properties

- ID

- name
- shortname
- SERVERS
- auth_domain

FACILITY Object

In Opsware SAS, a Facility is a collection of servers and the database that stores information about the environment in which those servers reside, all managed by a single Opsware Model Repository.

Properties

- ID
- name
- shortname
- subdomain
- status
- SERVERS

JOB Object

A job is any function in Opsware SAS that has been run, is running, or is scheduled to run.

Properties

- ID
- results
- username
- name
- end
- commands
- params
- start
- status

PACKAGE Object

A package is the collection of executables, configuration, or script files that are associated with an Opsware-installable application or program. In Opsware SAS, a package contains software package files registered in the Software Repository, including software for operating systems, applications (for example, BEA WebLogic, IBM WebSphere), databases, customer code, and software configuration information.

Properties

- ID
- name
- filename
- description
- notes
- date_created
- date_modified
- modified_by
- file_size
- wordpath
- created_by
- PLATFORM
- CUSTOMER
- reboot_on_install
- reboot_on_uninstall
- status

PLATFORM Object

A platform is any operating system supported by Opsware SAS.

Properties

- ID
- name

- shortname
- description
- architecture
- version
- SERVERS

ROLE Object

The ROLE object refers to the Way permissions role.

Properties

- ID
- name
- USERS

SERVER Object

A server in Opsware SAS is actually any specific hardware. Specific nodes are attached to servers that determine the specific software, configuration, and other server attributes.

Properties

- name
- hostname
- ip
- notes
- use
- stage
- CUSTOMER
- APPLICATIONS
- PLATFORM
- FACILITY
- cpu
- ram

- origin
- last_registration

SERVICELEVEL Object

Opsware SAS uses user-defined categories to group servers in an arbitrary way. For example, a user can group servers by functionality, tier, application, or ontogeny.

Properties

- ID
- name
- fullname
- description
- notes
- PLATFORM
- CUSTOMER
- PARENT
- SUBNODES
- SERVERS

TEMPLATE Object

Templates are used to install a set of (usually related) applications through a single invocation of a wizard. Templates and folders inherit all attachments of the folder they reside in. Inheritance is propagated from parent (folder) to child (template or folder) and to all children of children.

Properties

- ID
- name
- fullname
- description
- notes
- is_folder

- PLATFORM
- CUSTOMER
- PARENT
- SUBNODES

USER Object

A user is an individual with access to Opsware SAS. The level of access is provided by the assignment of user roles to individual users by the Opsware administrator.

Properties

- ID
- name
- ROLES
- JOBS

Appendix A: OCLI Syntax Tutorial

IN THIS APPENDIX

This Appendix describes how to use the OCLI Command line, and contains the following topics:

- OCLI Command Line
- OCLI Command Syntax

OCLI Command Line

The OCLI command line utility is intended as an alternative to using the Opsware Command Center. You can perform virtually every Opsware SAS function from the command line with this utility, with the exception of creating users and customers. These two functions must be performed from within the Opsware Command Center only, because of the business logic required to create them.

This OCLI Syntax Tutorial is intended to supplement the Command Reference and Object Properties information found in Chapter 2, “OCLI Commands”. Each command, and the properties of each Object used in commands is explained more fully there.

This introduction to the OCLI assumes that you have a copy of OCLI installed locally on a computer, and that the computer has the appropriate network configuration and the necessary certificate for communicating with the Opsware core.

If you are a Windows user, in order to follow these examples, `ocli.cmd` must be in the executable search path.

If you are a Unix user, the OCLI function must be defined by running the command:

```
eval `./ocli_setup.py`.
```

To test that your OCLI and all its components are installed correctly, execute the command:

```
% ocli
```

if everything is installed correctly, the system will return several screens of information about using OCLI.

To test that your OCLI is communicating properly with the Opsware core, enter these commands:

```
% ocli CUSTOMER any
--customer=0
```

OCLI Command Syntax

Each invocation of the OCLI utility can include parameters, a command, and some arguments. Most commands operate on objects within the Opsware object model, and have a syntax of the form:

```
ocli [parameters] [optional object parameters] command [command
arguments]
```

Simple Commands

Let's try some simple commands that will retrieve some information from Opsware SAS. This information is also available through the Opsware Command Center for browsing or additional detail.

In the following command, OCLI will retrieve from the Opsware core the ID of the customer called "Customer Independent". The results indicate that the customer has an ID of 0.

```
% ocli CUSTOMER "Customer Independent"
--customer=0
```

We could also enter a command to find out the ID for the customer called "any". (This is the short name for the customer, "Customer Independent".)

```
% ocli CUSTOMER any
--customer=0
```

Now that we know the ID for this customer, we can use it in subsequent commands. For example, let's retrieve information about the name, shortname, and the ID (which we already know).

```
% ocli --customer=0 get name
Customer Independent
% ocli --customer=0 get shortname
any
% ocli --customer=0 get id
0
```

Next, let's get a listing of the customers that are defined in Opsware SAS. We can do that with the `list` command.

```
% ocli list CUSTOMERS
--customer=0 --customer=9 --customer=15 --customer=100
```

Let's retrieve information about each of these customers.

```
% ocli --customer=0 get name
Customer Independent
% ocli --customer=9 get name
Not Assigned
% ocli --customer=15 get name
Opsware
% ocli --customer=100 get name
ACME Dynamite Corporation
```

Some properties of objects are other objects. For example, we will get a listing of servers that are assigned to the customer “Opsware”, and then retrieve the name for each server.

```
% ocli CUSTOMER opsware
--customer=15

% ocli --customer=15 get SERVERS
--server=60200 --server=10199 --server=20199

% ocli --server=60200 get name
styx.example.com

% ocli --server=10199 get name
oberon.example.com

% ocli --server=20199 get name
nile.example.com
```

There is much more information that can be obtained from Opsware SAS about a server. Here are some examples.

```
% ocli SERVER oberon.example.com
--server=10199

% ocli --server=10199 get id
10199

% ocli --server=10199 get ip
10.128.35.1

% ocli --server=10199 get use
PRODUCTION
```

```
% ocli --server=10199 get stage
LIVE

% ocli --server=10199 get origin
ASSIMILATED
```

If you want, you can combine the retrieval of several properties on the same line.

```
% ocli SERVER oberon.example.com
--server=10199
% ocli --server=10199 get ip use stage origin
10.128.35.1 PRODUCTION LIVE ASSIMILATED
```

Each server also has several properties that are also objects.

```
% ocli SERVER oberon.example.com
--server=10199
% ocli --server=10199 get CUSTOMER
--customer=15
% ocli --customer=15 get name
Opsware
% ocli --server=10199 get PLATFORM
--platform=960007
% ocli --platform=960007 get name
Red Hat Enterprise Linux AS 2.1
% ocli --server=10199 get FACILITY
--facility=200
% ocli --facility=200 get name
Test Core in VLAN35
```

Chaining Commands

You can chain many OCLI commands together, using the output of one command as the input for the next command in the chain. The syntax for that looks like this:

```
ocli command [parameters] | ocli command [parameters] | ocli
command [parameters] ...
```

In this simple example, we will pipe the output of the command to find a customer with a shortname of “any”, into the command to get the full name.

```
% ocli CUSTOMER any | ocli get name
Customer Independent
```

If you remember from previous examples, the command `ocli CUSTOMER any` will return the output `--customer=0`. This output is, in turn, used as input for the command `ocli get name`, saving us the trouble of typing `--customer=0` in the next command.

In the next example, we'll retrieve the same information that we got from a previous server example.

```
% ocli SERVER oberon.example.com | ocli get CUSTOMER | ocli get
name
Opsware
% ocli SERVER oberon.example.com | ocli get PLATFORM | ocli get
name
Red Hat Enterprise Linux AS 2.1
% ocli SERVER oberon.example.com | ocli get FACILITY | ocli get
name
Test Core in VLAN35
```

This yields the same results, with fewer commands to input.

Here is another recap of a previous example.

```
% ocli CUSTOMER opsware | ocli get SERVERS | ocli get name ip
styx.example.com 10.128.35.1
oberon.example.com 10.128.34.2
nile.example.com 10.128.34.1
```

In this example, we start by looking up the object for the customer “Opsware”. The output from this is piped to the command to look up all servers for the customer “Opsware.” The output from this command is piped to the final command, which looks up the name and IP address for each server. In essence we are, in a single command, asking the system to identify the host names and IP addresses of all servers that belong to the CUSTOMER “Opsware”.

```
% ocli CUSTOMER opsware
--customer=15
% ocli CUSTOMER opsware | ocli get SERVERS
--server=60200 --server=10199 --server=20199
% ocli CUSTOMER opsware | ocli get SERVERS | ocli get name ip
styx.example.com 10.128.35.1
oberon.example.com 10.128.34.2
nile.example.com 10.128.34.1
```

Finally, let's get the names of all customers known to Opsware SAS, and their IDs.

```
ocli list CUSTOMERS | ocli get id name
0 Customer Independent
7 Not Assigned
15 Opsware
100 ACME Dynamite Corporation
```

Working with Custom Attributes

Here are some simple commands to use with custom attributes. (This example uses the syntax for a Windows command interpreter (cmd.exe); please adjust the syntax for use on your specific platform.)

This example shows how to read a custom attribute on the current server.

```
C:\>ocli this | ocli read attr1
value1
```

This example shows how to read a custom attribute from another server.

```
C:\>ocli SERVER foo.exmaple.com | ocli read attr2
value2
```

This example shows how to read a custom attribute from the current server's facility.

```
C:\>ocli this | ocli get FACILITY | ocli read attr3
value3
```

This example shows how to read a custom attribute from the current server's customer.

```
C:\>ocli this | ocli get CUSTOMER | ocli read attr4
value4
```

This example shows how to read many custom attributes at once from the current server's customer.

```
C:\>ocli this | ocli get CUSTOMER | ocli read attr5 attr6 attr7
value5 value6 value7
```

This example shows how to write a custom attribute to the current server.

```
C:\>ocli this
--server=32402099
C:\>set this=--server=32402099
C:\>echo Not hungry any more | ocli %this% write current_state
C:\>ocli this | ocli read current_state
full of values
```

Set Operations

The OCLI provides a number of set operations to simplify certain tasks. In the following examples, we will be working with applications and the servers that are attached to these applications. (We will also be showing these examples with Unix bourne shell syntax; please adjust the syntax for use on for your specific platform.)

We will start by creating environment variables that represent two applications, which we will use to demonstrate set operations.

```
% app1=`omcd APPLICATION "Operating System Extras / RedHat 3.0
ES / TripWire / 4.0.2"`; export app1

% app2=`omcd APPLICATION "Operating System Extras / RedHat 3.0
ES / TripWire / 4.0.3"`; export app2

% echo $app1 $app2
--application=3502099 --application=3503099
```

Next, let's see what servers are attached to each of these nodes, and assign the results to environment variables for further use.

```
% ocli get $app1 SERVERS
--server=6510199 --server=6560199 --server=6790199
--server=6800199 --server=7030199 --server=7130199

% ocli get $app2 SERVERS
--server=6560199 --server=6560199 --server=6780199
--server=6800199 --server=6810199 --server=7130199
--server=7230199

% serverlist1=`ocli get $app1 SERVERS`; export serverlist1

% serverlist2=`ocli get $app2 SERVERS`; export serverlist2
```

If we wanted to see servers that are attached to either application node, we could use the following example.

```
ocli $serverlist1 $serverlist2 union
--server=7030199 --server=7230199 --server=7130199
--server=6800199 --server=6560199 --server=6810199
--server=6510199 --server=6780199 --server=6790199
ocli $serverlist1 $serverlist2 union | ocli get id name
7030199 host6402.example.com
7230199 host6501.example.com
7130199 host6405.example.com
6800199 host6302.example.com
6560199 host6202.example.com
6810199 host6303.example.com
6510199 host6201.example.com
6780199 host6302.example.com
6790199 host6302.example.com
```

Now we will retrieve a list of only those servers that are attached to both nodes.

```
ocli $serverlist1 $serverlist2 intersection | ocli get id name
7130199 host6405.example.com
6800199 host6302.example.com
6560199 host6202.example.com
```

Now we will retrieve a list of only those servers that are attached to one node or the other, but not both.

```
ocli $serverlist1 $serverlist2 disjunction | ocli get id name
7030199 host6402.example.com
7230199 host6501.example.com
6810199 host6303.example.com
6510199 host6201.example.com
6780199 host6302.example.com
6790199 host6302.example.com
```

Finally, we will get a list of servers that are members of serverlist1, but not serverlist2.

```
ocli $serverlist1 $serverlist2 disjunction | ocli $serverlist1
intersection ocli get id name
6510199 host6201.example.com
6790199 host6302.example.com
7030199 host6402.example.com
```