Coreutils Library

Table of contents

1. Welcome to "coreutils"

What is it?

A set of utilities inspired after the GNU coreutils.

Why use it?

- Cross platform utilities.

Getting started

You can start using the this library by following the steps of the documentation pages listed below:

- Install: Describes how to download and install the library
- Configure: Describes how to configure the library for deployment
- Run: Explains how to run the commands.

2. Install

2.1. Install

Overview

This document describes the installation steps necessary to use the library.

2.1.1. Step #1: Install the CTL software

This library assumes you have installed the latest stable release of the CTL software on a a designated server host and one or more client hosts. Refer to the general installation procedures for more info.

2.1.2. Step #2: Download the library archive

Binary distributions of the library can be found in the "File Releases" section of the ModuleForge Download page on Sourceforge. The package is called "coreutils" and will be named something like: coreutils-extension-*x.y*.jar where "x.y" denote the version.

It is always suggested to download the latest release.

2.1.3. Step #3: Install the library archive

The library archive must be installed into your CTL instance. You will use the extension installer for this procedure.

Use the `ext-setup` command to install the archive. Do this on the server host to ensure the modules within the archive will be published to the repository.

```
ext-setup -f /path/to/coreutils-extension-x.y.jar -R
```

2.1.4. Installing from source

1. Create the workspace

```
mkdir $CTL_BASE/src; cd $CTL_BASE
```

2. Checkout files

```
svn co
https://moduleforge.svn.sourceforge.net/svnroot/moduleforge/coreutils/trunk/src
```

3. Build the library as an extension

```
ctl -m ProjectBuilder -c build-library -- -archive extension -name
coreutils
```

This will generate a a new file `$CTL_BASE/target/coreutils-extension.jar`

4. Install the extension

```
ext-setup -f $CTL_BASE/target/coreutils-extension.jar
```

2.1.5. Updating project depots

The extension is installed at the framework level but does not automatically get installed into your project depots.

Run the `depot-setup` command:

```
depot-setup -p default
```

Once the library has been installed into your chosen project the next step is to configure these modules for use.

Next: Configure #

3. Configure

3.1. Configure

Overview

This document describes how to configure a project to use the coreutils library.

Note:

Be sure you have already installed the CTL software, and loaded the library archive. See the Install page for instructions.

3.1.1. Step #1: Edit deployments.properties

With the library archive installed you are ready to specify where its modules will be used. As described in the general CTL project setup page, this is done in the `deployments.properties` file.

Each project has a `deployments.properties` file centrally maintained in the repository. Use a text editor and register where you want the modules deployed.

```
### module deployments format:
#
# module-deployment.project.module = node1,node2,...,node3
#
# example: stipulate that for "default" all the
#          "coreutils" modules be deployed to node1,node2,node3
#
module-deployment.default.fileutil = node1,node2,node3
module-deployment.default.netutil = node1,node2,node3
module-deployment.default.shellutil = node1,node2,node3
module-deployment.default.textutil = node1,node2,node3
```

3.1.2. Step #2: Install to project depots

Next on the client machines where your project is deployed run:

```
depot-setup -p project -a install
```

3.1.3. Optional Step: Check-in deployments.properties file

It is considered best practice to maintain the CTL configuration files in a source code repository.

Next: Run #

4. Run

4.1. Run

Overview

This document describes how to run commands in the coreutils library.

4.1.1. Execute via 'ctl'

In depth descriptions of the Content library commands can be found in the Reference section of the documentation.

Next: Reference #

5. Reference

5.1. Reference

Overview

This section is useful to developers interested how the library works and users interested in knowing all the command syntax offered by the modules in this library.

All the attributes and commands are defined in the Type Reference section of the documentation. There you will find a document page for each type.

Next: Type Reference #

5.2. Types

5.2.1. Type Reference

- coretests: Test suite for coreutils code
- fileutil: various file utilities
- netutil: various network utilities
- shellutil: various shell utilities
- textutil: various text utilities

5.2.2. fileutil

5.2.2.1. Overview

fileutil: *various file utilities*

5.2.2.2. Design

Super Type
Managed-Entity

Role	**Abstract**. (Objects cannot be created.)
Instance Names	**Unique**
Notification	**false**
Template Directory	
Data View	Children, proximity: **1**
Logger Name	fileutil

5.2.2.3. Constraints

None

5.2.2.4. Attributes

None

5.2.2.5. Commands

Note:
Commandline options displayed in square brackets "[]" are optional. If an option expects arguments, then angle brackets are shown after the option "<>". Any default value is shown within the brackets.

available

test if file available

static: This command can be run outside of an object context.

Usage

```
available [-failonerror] -file <>
```

Options

Option	Description
`failonerror`	*error the command if test fails*
`file`	*file name*

executable

test if file executable

static: This command can be run outside of an object context.

Usage

```
executable [-failonerror] -file <>
```

Options

Option	Description
`failonerror`	*error the command if test fails*
`file`	*file name*

writeable

test if file writeable

static: This command can be run outside of an object context.

Usage

```
writeable [-failonerror] -file <>
```

Options

Option	Description
`failonerror`	*error the command if test fails*
`file`	*file name*

readable

test if file readable

static: This command can be run outside of an object context.

Usage

```
readable [-failonerror] -file <>
```

Options

Option	Description
`failonerror`	*error the command if test fails*
`file`	*file name*

older

test if file older than other file

static: This command can be run outside of an object context.

Usage

```
older [-failonerror] -file <> -target <>
```

Options

Option	Description
failonerror	*error the command if test fails*
file	*file name*
target	*target file name*

newer

test if file newer than other file

static: This command can be run outside of an object context.

Usage

```
newer [-failonerror] -file <> -target <>
```

Options

Option	Description
failonerror	*error the command if test fails*
file	*file name*
target	*target file name*

ls

list directory

static: This command can be run outside of an object context.

Usage

```
ls -dir <> [-recursive]
```

Options

Option	Description
dir	*directory name*
recursive	*list recursively*

mkdir

make directory

static: This command can be run outside of an object context.

Usage

```
mkdir -dir <>
```

Options

Option	Description
`dir`	*directory name*

rmdir

remove directory

static: This command can be run outside of an object context.

Usage

```
rmdir -dir <> [-failonerror]
```

Options

Option	Description
`dir`	*directory name*
`failonerror`	*if true fail on error*

copy

copy file

static: This command can be run outside of an object context.

Usage

```
copy [-backup] -file <> [-overwrite] [-suffix <.backup>]
-target <>
```

Options

Option	Description
`backup`	*if present make a backup if the target file exists and -overwrite is specified*
`file`	*file name*

`overwrite`	*if present overwrite any existing file*
`suffix`	*suffix of backup file*
`target`	*target file name*

move

move file

static: This command can be run outside of an object context.

Usage

```
move -file <> -target <>
```

Options

Option	Description
`file`	*file name*
`target`	*target file name*

remove

remove file

static: This command can be run outside of an object context.

Usage

```
remove [-failonerror] -file <>
```

Options

Option	Description
`failonerror`	*error the command if test fails*
`file`	*file name*

link

make a link

static: This command can be run outside of an object context.

Usage

```
link -file <> -target <>
```

Options

Option	Description
`file`	*link file to be created*
`target`	*file for link target*

touch

touch a file

static: This command can be run outside of an object context.

Usage

```
touch [-datetime <>] -file <>
```

Options

Option	Description
`datetime`	*specifies new mod time of the file*
`file`	*file name*

5.2.3. netutil

5.2.3.1. Overview

netutil: *various network utilities*

5.2.3.2. Design

Super Type

Managed-Entity

Role	**Abstract**. (Objects cannot be created.)
Instance Names	**Unique**
Notification	**false**
Template Directory	
Data View	Children, proximity: **1**
Logger Name	netutil

5.2.3.3. Constraints

None

5.2.3.4. Attributes

None

5.2.3.5. Commands

Note:
Commandline options displayed in square brackets "[]" are optional. If an option expects arguments, then angle brackets are shown after the option "<>". Any default value is shown within the brackets.

listening

hecks for the existence of a TCP/IP listener at the specified host and port.

static: This command can be run outside of an object context.

Usage

```
listening -port <> [-server <localhost>]
```

Options

Option	Description
`port`	*The port number to connect to.*
`server`	*The DNS name or IP address of the server.*

reachable

test if server is reachable

static: This command can be run outside of an object context.

Usage

```
reachable -server <> [-timeout <10>]
```

Options

Option	Description
`server`	*The DNS name or IP address of the server.*
`timeout`	*count*

traceroute

print the route packets take to network host

static: This command can be run outside of an object context.

Usage

```
traceroute -count <3> -server <>
```

Options

Option	Description
`count`	*count*
`server`	*The DNS name or IP address of the server.*

5.2.4. shellutil

5.2.4.1. Overview

shellutil: *various shell utilities*

This module contains a set of commands useful for managing system processes.

5.2.4.2. Design

Super Type

Managed-Entity

Role	**Abstract**. (Objects cannot be created.)
Instance Names	**Unique**
Notification	**false**
Template Directory	
Data View	Children, proximity: **1**
Logger Name	shellutil

5.2.4.3. Constraints

None

5.2.4.4. Attributes

None

5.2.4.5. Commands

Note:
Commandline options displayed in square brackets "[]" are optional. If an option expects arguments, then angle brackets are shown after the option "<>". Any default value is shown within the brackets.

env

get environment info

The `env` command looks up values of environment variables. Use the `-key` option to specify a particular key to lookup. If no `key` option is specified all environment variables are printed.

Format: Uses YAML to print lookup results.

Example: Print the value of the CTL_BASE variable

```
$ ctl -m shellutil -c env -- -key CTL_BASE
---
CTL_BASE: /ctier/ctl
```

static: This command can be run outside of an object context.

Usage

```
env [-key <.*>] [-output <>]
```

Options

Option	Description
`key`	*env key*
`output`	*output file*

exec

executes a command

static: This command can be run outside of an object context.

Usage

```
exec [-argline <>] -executable <> [-failonerror <false>]
[-os <>] [-output <>] [-script <>] [-scriptfile <>]
[-timeout <>]
```

Options

Option	Description
argline	*arguments for execututable*
executable	*executable path*
failonerror	*fail if there is an error*
os	*list of Operating Systems on which the command may be executed.*
output	*Name of a file to which to write the output. If the error stream is not also redirected to a file or property, it will appear in this output.*
script	*script to execute*
scriptfile	*scriptfile to execute*
timeout	*Stop the command if it doesn't finish within the specified time*

kill

kill process

static: This command can be run outside of an object context.

Usage

```
kill -pid <> [-sig <TERM>]
```

Options

Option	Description
pid	*process id*
sig	*signal to send*

pkill

kill named process

static: This command can be run outside of an object context.

Usage

```
pkill [-pname <>] [-sig <TERM>]
```

Options

Option	Description
pname	*process name*
sig	*signal to send*

ps

get process info

Obtains and prints process table information.

The following pieces of information are shown for each running process:

- user: user name
- command: command and arguments
- pcpu: percentage cpu usage
- pmem: percentage memory usage
- state: symbolic process state
- pid: process ID

Example: Show bash processes

```
$ ctl -m shellutil -c ps -- -pname bash
---
user: alexh
command: bash
pmem: "0.0"
args: "[-login]"
state: S
pid: "57085"
pcpu: "0.0"
```

static: This command can be run outside of an object context.

Usage

```
ps [-format <yaml>] [-output <>] [-pname <.*>]
```

Options

Option	Description
format	*output format. (plain, yaml)* Display the result in the spcified format. If "plain" is specified, results are displayed in their native ps command format. If "yaml" is specified then each process is displayed in a key/value pair format.
output	*output file*

	Save the results to the output file. If no `-output` option is specified results are printed to the terminal.
`pname`	*process name* Filter process info to those matching specified process name. The value for the `-pname` argument can also be a regular expression. If the argument is unspecified then all processes are listed.

whoami

print the user name

static: This command can be run outside of an object context.

Usage
`whoami`

Options

Option	Description

5.2.5. textutil

5.2.5.1. Overview

textutil: *various text utilities*

5.2.5.2. Design

Super Type
Managed-Entity

Role	**Abstract**. (Objects cannot be created.)
Instance Names	**Unique**
Notification	**false**
Template Directory	
Data View	Children, proximity: **1**
Logger Name	textutil

5.2.5.3. Constraints

None

5.2.5.4. Attributes

None

5.2.5.5. Commands

Note:
Commandline options displayed in square brackets "[]" are optional. If an option expects arguments, then angle brackets are shown after the option "<>". Any default value is shown within the brackets.

cat

concatenate the file

static: This command can be run outside of an object context.

Usage

```
cat [-dir <>] -files <>
```

Options

Option	Description
`dir`	*dir name*
`files`	*file names*

checksum

checksum a file

static: This command can be run outside of an object context.

Usage

```
checksum [-algorithm <MD5>] [-createfile] -file <>
```

Options

Option	Description
`algorithm`	*Specifies the algorithm to be used to compute the checksum. Defaults to MD5. Other popular algorithms like SHA may be used as well.*
`createfile`	*save the checksum to file*

`file`	*file name*

head

display first lines of a file

static: This command can be run outside of an object context.

Usage

```
head [-count <10>] -file <>
```

Options

Option	Description
`count`	*line count*
`file`	*file name*

nl

show file content with line numbers

static: This command can be run outside of an object context.

Usage

```
nl -file <>
```

Options

Option	Description
`file`	*file name*

sort

sort file

This is a placeholder implementation. It currently shells out if it is a unix OS

static: This command can be run outside of an object context.

Usage

```
sort -file <>
```

Options

Option	Description
`file`	*file name*

tail

display last lines of a file

static: This command can be run outside of an object context.

Usage

```
tail [-count <10>] -file <>
```

Options

Option	Description
`count`	*line count*
`file`	*file name*

uniq

uniq file

This is a placeholder implementation. It currently shells out if it is a unix OS

static: This command can be run outside of an object context.

Usage

```
uniq -file <>
```

Options

Option	Description
`file`	*file name*

wc

ord, line, character, and byte count

This is a placeholder implementation. It currently shells out if it is a unix OS

static: This command can be run outside of an object context.

Usage

```
wc [-bytes] [-chars] -file <> [-lines] [-words]
```

Options

Option	Description
`bytes`	*number of bytes in file*

`chars`	*number of chars in file*
`file`	*file name*
`lines`	*number of lines in file*
`words`	*number of words in file*

6. All