Copyright © 2002 – 2014 Electric Cloud, Inc. All rights reserved.

Published 7/28/2014

Electric Cloud® believes the information in this publication is accurate as of its publication date. The information is subject to change without notice and does not represent a commitment from the vendor.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED “AS IS.” ELECTRIC CLOUD, INCORPORATED MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any ELECTRIC CLOUD software described in this publication requires an applicable software license.

Copyright protection includes all forms and matters of copyrightable material and information now allowed by statutory or judicial law or hereinafter granted, including without limitation, material generated from software programs displayed on the screen such as icons, screen display appearance, and so on.

The software and/or databases described in this document are furnished under a license agreement or nondisclosure agreement. The software and/or databases may be used or copied only in accordance with terms of the agreement. It is against the law to copy the software on any medium except as specifically allowed in the license or nondisclosure agreement.

Trademarks

Electric Cloud, ElectricAccelerator, ElectricCommander, ElectricInsight, and Electric Make are registered trademarks or trademarks of Electric Cloud, Incorporated.

Electric Cloud products—ElectricAccelerator, ElectricCommander, ElectricInsight, and Electric Make—are commonly referred to by their "short names”—Accelerator, Commander, Insight, and eMake—throughout various types of Electric Cloud product-specific documentation.

Other product names mentioned in this guide may be trademarks or registered trademarks of their respective owners and are hereby acknowledged.
# ElectricFlow in the Electric Cloud Environment

# How to Use the ElectricFlow API

## Using etool

### Logging in

### Global Arguments (optional)

### Passing Lists as Arguments

## Using Perl

### Perl API structure

## Common Global Options

## The Batch API

### Using the Batch API

### Installing Commander Perl modules into Your Perl Distribution

### Installing Perl Modules into the Commander Perl Distribution

### When Upgrading Commander

## API commands - ACL Management

## API commands - Applications

## API commands - Application Tier

## API commands - Artifact Management

## API commands - Component

## API Commands - Credential Management

## API Commands - Database Configuration

## API Commands - Directory Provider Management

## API Commands - Email Configuration Management

## API Commands - Email Notifier Management

## API Commands - Environment Requests

  - createEnvironment
  - createEnvironmentInventoryItem
  - deleteEnvironment
  - deleteEnvironmentInventoryItem
  - getEnvironment
  - getEnvironments
  - getEnvironmentApplications
  - getEnvironmentInventory
  - getEnvironmentInventoryItem
  - getEnvironmentInventoryItems
  - modifyEnvironment
  - modifyEnvironmentInventoryItem

## API Commands - Environment Tier

  - createEnvironmentTier
  - deleteEnvironmentTier
getEnvironmentTier ........................................................................ 138
getEnvironmentTiers ..................................................................... 138
modifyEnvironmentTier ................................................................. 139

API Commands - Gateways/Zones Management .................................... 141

API Commands - Job Management .................................................. 150

External Job APIs ......................................................................... 166

API Commands - Parameter Management ......................................... 181

API Commands - Plugin Management ............................................... 199

API Commands - Procedure Management ......................................... 205

API Commands - Process .................................................................. 224
createProcess ................................................................................. 224
deleteProcess ................................................................................ 225
getProcess ..................................................................................... 226
getProcesses .................................................................................. 227
modifyProcess ................................................................................ 228
runProcess ..................................................................................... 230

API Commands - Process Dependency ............................................. 232
createProcessDependency ............................................................... 232
deleteProcessDependency .............................................................. 233
getProcessDependencies ................................................................. 235
modifyProcessDependency .............................................................. 236

API Commands - Process Step ......................................................... 238
createProcessStep .......................................................................... 238
deleteProcessStep ......................................................................... 240
getProcessStep .............................................................................. 241
getProcessSteps ............................................................................ 243
modifyProcessStep ......................................................................... 244

API Commands - Project Management ............................................ 247

API Commands - Property Management .......................................... 252

API Commands - Resource Management .......................................... 283

API Commands - Schedule Management .......................................... 302

API Commands - Server Management ............................................. 309

API Commands - Tier Map .............................................................. 317
createTierMap ................................................................................ 317
deleteTierMap ............................................................................... 318
deleteTierMapping .......................................................................... 319
getTierMaps .................................................................................. 320
modifyTierMap ............................................................................... 321

API Commands - User/Group Management ..................................... 323

API Commands - Workflow Management ......................................... 335
ElectricFlow in the Electric Cloud Environment

ElectricFlow is a complete end-to-end web-based software deployment solution. ElectricFlow automates standard build, test, deploy, and release processes across your enterprise. You can select the components of the working applications in your software environment.

You use the ElectricCommander platform tools and operations to access the API commands created for ElectricFlow. For example, `ectool getApplications` returns summary information for a list of applications in a project. It does not matter if you are using the application objects in ElectricFlow or ElectricCommander.

**IMPORTANT:** Currently all the API commands created for ElectricFlow are executed in a "Default" ElectricCommander project.
How to Use the ElectricFlow API

ElectricFlow features can be accessed in two ways:

- The most common access is through the web interface, which displays screens to create projects, procedures, and steps; launch jobs; and manage all administration tasks.
- The second access method is the Commander API. The API can be used from a command-line, including a shell script, or a batch file. Any operation you can perform on the web interface, you can perform using the API because they both rely on the same interface to the ElectricCommander server.

The Commander API supports ectool and ec-perl (or Perl) commands:

- **ectool** is a command-line tool developed to script ElectricFlow operations.
- **ec-perl** is delivered as a Perl package during ElectricFlow installation, or you can use any Perl of your choice.

Because ectool and ec-perl can work together, this section describes Perl and ectool usage and differences.

- **Using ectool**
- **Using ec-perl**
- **Common global options**
- **The Batch API**
- **Installing Commander Perl modules into your Perl distribution**
- **Installing Perl modules into the Commander Perl distribution**

**Using ectool**

*ectool* is a command-line application that provides operational control over the ElectricFlow system.

ectool supports a large collection of commands, each of which translates to a message sent to the ElectricCommander server.

For example, `ectool getProjects` returns information about all projects defined in the server.

- `ectool --help` displays a summary of all commands and other command-line options.
- For information about a particular command, use `--help` followed by the command name. For example, `ectool --help modifyStep` returns information about the `modifyStep` command.

**Logging in**

If you use ectool outside of a job, you *must* invoke the `ectool login` command to login to the server. After logging in, ectool saves information about the login session for use in future ectool invocations. If you run ectool as part of a ElectricFlow job, you do not need to log in—ectool uses the login session (and credentials) for that job.

To log in to a specific server, see the example below, which includes the server name, user name, and password.

Login example:

    ectool --server bldg1server login "Ellen Ernst" "ee123"

General syntax for ectool command usage:
ectool [global argument] <command> <positional arguments> [named arguments]

**Global Arguments (optional)**

See the Common global options section for more information.

**Passing Lists as Arguments**

Some API commands include arguments that expect a list of values. Two list forms: value lists and name/value pairs. The syntax to specify a list depends on whether you are using ectool or ec-perl.

**For ectool**

- **value list** - each value is specified as a separate argument on the command line
  
  Example:
  
  `ectool addUsersToGroup group1 --userNames user1 user2 user3`

- **name/value pairs** - each pair is specified as a separate argument in the form `name=value`
  
  Example:
  
  `ectool runProcedure proj1 --procedureName procl --actualParameter parm1=value1 parm2=value2`

**For ec-perl**

- **value list** - the argument value is a reference to an array of values
  
  Example:
  
  ```
  $cmdr->addUsersToGroup({
    groupName => 'group1',
    userName => ['user1', 'user2'])
  ```

- **name/value pairs** - the argument value is a reference to an array of hash references. Each hash contains a pair of entries, one for the name and one for the value. The hash keys depend on the specific API.
  
  Example:
  
  ```
  $cmdr->runProcedure({
    projectName => 'proj1',
    procedureName => 'procl',
    actualParameter => [{
      actualParameterName => 'parm1',
      value => 'value1'},
      { actualParameterName => 'parm2',
      value => 'value2'}])
  ```

**Using Perl**

When ElectricFlow is installed—Server, Agent, or Tools (using the express or advanced installation type)—a copy of Perl is installed. This Perl is pre-configured with all the packages you need to run the Commander Perl API. Commander does not, however, automatically add this version of Perl to your path because:

- We did not want the ElectricFlow installation to interfere with existing scripts you may run, which are dependent on finding another copy of Perl you already use.

- Some special environment variables need to be set before calling Perl.

Both of these issues are addressed with a small wrapper program called ec-perl. The wrapper is installed as part of ElectricFlow, and it is in a directory that is added to your path. When the ec-perl wrapper runs, it sets up the environment, finds, and calls the Commander copy of Perl, passing all of its parameters to Perl.

To run ec-perl from a command line (or in a ElectricFlow step) simply enter:

`ec-perl yourPerlOptions yourPerlScript.pl`
The Perl script can include API calls to ElectricFlow with no other special handling required.

Another way to write Perl scripts: For an ElectricFlow step, enter the Perl script directly into the "Command" field, and set the "Shell" field to ec-perl. The Commander-installed Perl is used to process the Perl script.

You can develop Perl scripts to access the Perl API directly. Because ectool uses the Perl API to execute its commands, any ectool command you can execute can be executed using the Perl API. If you are writing (or currently using) a script that makes tens or hundreds of calls, the Perl API provides a significant performance improvement over ectool.

The Perl API is delivered as a collection of Perl packages pre-installed in a Perl 5.8 distribution. The main API package is called ElectricCommander.

**Perl API structure**

The Perl API has the same four elements as ectool, but the way these elements are specified is quite different.

**Specifying global options**

To use the Commander Perl API, you must first create an object. Global arguments are specified at the time the object is created. These arguments are passed as members of an anonymous hash reference, as shown in the following example:

```perl
use ElectricCommander;
$cmdr = ElectricCommander->new({
    server => "vm-xpsp2",
    port   => "8000",
    securePort => "8443",
    debug  => "1",
});
```

In the example above, port options are not really necessary because they specify default values. When you want to specify the server name only, you can use the "shorthand" form:

```perl
use ElectricCommander;
$cmdr = ElectricCommander->new("vm-xpsp2");
```

An even simpler form can be used if you call the Perl API from a script running as part of an ElectricFlow job step. In this case, the Commander package sets the server name based on the environment variable, COMMANDER_SERVER, set by the Commander agent.

```perl
use ElectricCommander;
$cmdr = ElectricCommander->new();
```

To see a complete list of global commands you can use with Perl, click here.

**Note:** If your script uses International characters (non-ascii), add the following block to the top of your ec-perl command block:

```perl
use utf8;
ElectricCommander::initEncodings();
```

**Specifying subcommands**

For each subcommand, there is a corresponding Commander object function.

For example, to retrieve a list of jobs, use

```perl
$cmdr->getJobs();
```

**Specifying arguments**
Most subcommands expect one or more arguments. Arguments are specified as key value pairs in a hash ref passed as the final argument to the subcommand. Additionally, as a convenience, some arguments may be specified as positional arguments prior to the options hash ref.

For example, `setProperty` has two positional arguments, `propertyName` and `value`, as well as an optional `jobId` argument that can be specified in either of the following forms:

```perl
$cmdr->setProperty("/projects/test/buildNumber", "22",
    {jobId => $jobId});
```

or

```perl
$cmdr->setProperty({
    propertyName => "/projects/test/buildNumber",
    value => "22",
    jobId => $jobId });
```

Handling return values

Every function to the object returns an object of type `XML::XPath`. This is an object that returns a parsed representation of the ElectricFlow returned XML block. See documentation on CPAN for more information.

```perl
$xpath = $cmdr->setProperty("filename", "temp.xml");
print "Return data from Commander:\n".
    $xpath->findnodes_as_string ("/") . "\n";
```

Error handling

If a function call to the ElectricCommander object encounters an error, by default, it "dies" inside Perl and prints an error message. If you want to handle errors yourself and continue processing, you must set a flag to disable internal error handling and handle the error in your code.

For example:

```perl
$cmdr->abortOnError(0);
$xpath = $cmdr->getResource("NonExistent Resource");
if ($xpath) {
    my $code = $xpath->findvalue('//code')->value();
    if ($code ne "") {
        my $mesg = $xpath->findvalue('//message');
        print "Returned code is "$code\n$mesg\n";
        exit 1;
    }
}
```

An alternative to using the `abortOnError` flag:

```perl
eval {$cmdr->get...};
if ($@) {
    print "bad stuff: $@";
    exit 1;
}
```

Specifying a named object

Any API argument that refers to a named object (for example, `eventName`, `procedureName`) performs property reference expansion before looking in the database for the object. This process allows constructs like the following to work without making two separate server requests:

```perl
$cmdr->getProject ("${[server/defaultProject]}")
```

Property reference expansion for names occurs in the global context, so context-relative shortcuts like "myProject" are not available.
# Common Global Options

Global arguments can be used alone or in conjunction with other commands. These arguments are used to control communication with the server and can be used with the ectool or ec-perl API.

<table>
<thead>
<tr>
<th>Global Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help</td>
<td>Display an online version of ectool commands with a short description. Displays command information if followed by a command name.</td>
</tr>
<tr>
<td>--version</td>
<td>Display the ectool version number.</td>
</tr>
<tr>
<td>--server &lt;hostname&gt;</td>
<td>ElectricCommander server address. Defaults to the COMANDER_SERVER environment variable. If this variable does not exist, the default is to localhost. Note: If you are using multiple servers, Electric Cloud recommends using the server option to ensure the correct server is specified for your task. For example, if you are using the import API, the server option may be particularly important. Do not use in a step context: Electric Cloud recommends that steps running ectool or Perl scripts should never provide the server option if the intention is to communicate with the server that launched the step. If the intention is to communicate with a different server, this agent must be a registered, enabled resource in the second server. Thus, that server will ping the agent, and the agent will learn how to communicate with that server. In a step context, ectool and the Perl API proxy server requests through the step's agent. If the agent does not recognize the provided server-name, it rejects the request. ectool / Perl API retry the operation because at some point the server should ping the agent, and then the agent will have learned how to communicate with the server. Generally, the issue is that the server publicizes its name as a fully-qualified domain name and ectool / Perl API issue requests with a simple-name for the server. This can happen if the step explicitly states which server it is connecting to. Fix your steps that invoke ectool so they no longer include the server-name, and ectool will default to the server-name that the server provided.</td>
</tr>
<tr>
<td>--port &lt;port&gt;</td>
<td>HTTP listener port on the ElectricCommander server. Defaults to port 8000.</td>
</tr>
<tr>
<td>--securePort &lt;secureport&gt;</td>
<td>HTTPS listener port on the ElectricCommander server. Defaults to port 8443.</td>
</tr>
<tr>
<td>--secure &lt;0</td>
<td>1&gt;</td>
</tr>
<tr>
<td>Global Arguments</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>--timeout &lt;s&gt;</td>
<td>An API call waits for a response from the server for a specified amount of time. Timeout for server communication defaults to 180 seconds (3 minutes) if no other time is specified. After the timeout, the API call stops waiting for a response, but the server continues to process the command.</td>
</tr>
<tr>
<td>--retryTimeout &lt;s&gt;</td>
<td>This is a separate timer, independent of the retry flag, and used to control Commander’s automatic error recovery. When the API is unable to contact the Commander server, it will keep trying to contact the server for this length of time. When the API is called from inside a step, it defaults to 24 hours.</td>
</tr>
<tr>
<td>--retry &lt;0</td>
<td>1&gt;</td>
</tr>
<tr>
<td>--user &lt;username&gt;</td>
<td>Use the session associated with the user. Defaults to the user who last logged in.</td>
</tr>
<tr>
<td>--service &lt;spn&gt;</td>
<td>Specify the service principal name to use for Kerberos. Defaults to <a href="mailto:HTTP@host.domain">HTTP@host.domain</a>.</td>
</tr>
<tr>
<td>--setMax &lt;0</td>
<td>1&gt;</td>
</tr>
<tr>
<td>encoding &lt;charEncoding&gt;</td>
<td>Use the specified encoding for input/output. For example, for charEncoding, supply UTF-8, cp 437, and so on. Default is autodetected.</td>
</tr>
<tr>
<td>--dryrun</td>
<td>Displays session information and the request that would be sent, without communicating with the server. If a subcommand is specified, the server request that would be sent is displayed. This option can also be used to change the default user/server value by specifying the --user or --server options.</td>
</tr>
<tr>
<td>--silent</td>
<td>Suppresses printing the result. For example: ectool --silent createResource foo will not print the resource name, agent state, any modify information, create time, owner, port, or any other information otherwise displayed when you create a resource.</td>
</tr>
<tr>
<td>--valueOf</td>
<td>This option can return the value of a unique element. Because many ectool APIs return an XML result, it is inconvenient to use ectool in shell scripts and makefiles where you might want a piece of the ectool result to incorporate into some other logic. Using the --valueOf &lt;path&gt; option evaluates the XML result and emits the value of that node to satisfy such use cases. For example: $ ectool --valueOf '//version' getServerStatus returns only &quot;4.1.0.48418&quot;.</td>
</tr>
<tr>
<td>--format &lt;format&gt;</td>
<td>Specifies the response format. Must be one of 'xml' or 'json'. Defaults to 'xml'. For example, you might specify: ectool --format json setProperty summary hello</td>
</tr>
<tr>
<td>--ignoreEnvironment</td>
<td>Force ectool to ignore COMMANDER_ENV variables.</td>
</tr>
</tbody>
</table>
The Batch API

The Perl API supports a batch operation mode that allows you to send multiple API requests in a single "envelope", which has several advantages over standard, individual API calls in some situations. For example, you could use the batch API when you need to set 10 or even 100 property values.

The batch API reduces "round-trip" transmissions. All SetProperty requests can be sent in a single envelope. You can choose an option that changes all properties in a single database transaction in the server. This means changes are made using an "all or none" approach. If one change fails, they all fail, which allows you to keep your data in a consistent state. When you make a large number of requests in one envelope, the single database transaction option provides much better performance.

Using the Batch API

To use the batch API, first create a object as you would for a standard API. From your newly created object, create a batch object using the newBatch method. The newBatch method takes a single argument, which is the "request processor mode". This argument tells the server how to process multiple requests. There are three "request processor modes":

1. serial - each request in the envelope is processed serially, each in its own transaction.
2. parallel - each request in the envelope is processed in parallel, each in its own transaction.
3. single - each request in the envelope is processed serially, all in the same transaction.

Specifying serial, parallel, or single is optional. If you do not specify an option, the server determines the best mode to use, based on the requests in the envelope.

Example - creating a batch object:

use ElectricCommander;
my $cmdr = ElectricCommander;
# Create the batch API object
my $batch = $cmdr->newBatch("parallel");

The batch object supports all the same calls as the standard API. The result of each call is a numeric requestId that can be used to locate a response from an individual request within the batch.

Example - creating multiple requests in a batch:

# Create multiple requests
my @reqIds = {
    $batch->setProperty("/myJob/p1", 99),
    $batch->incrementProperty("/myJob/p2");
};

After the batch is created, submit it to the server for processing. The return from the submit() call is an XPath object that represents an XML document containing the responses for all of the API requests.

Example - submitting the batch:

# Submit all the requests in a single envelope
$batch->submit();

Sample response from this example:

<responses xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
    xsi:
        version="2.1" dispatchId=1680
    <response requestId="1">
        <property>
To extract information from the response to a request, use standard XPath syntax, and supply the requestID returned by that specific API call to either the find or findvalue functions on the batch object.

Example - extracting response information:

```perl
# Extract the value from the "increment" request
my $value = $batch->findvalue($reqIds[0], 'property/value');
print "New value is $value\n";
```

Single-transaction batch processing can continue after errors if you supply an ignoreErrors attribute in the request and/or requests elements. The ignoreErrors value is evaluated as a regular expression against any error codes from the batch. If the expression matches, an error will not cause the batch to fail.

There are two ways to specify ignoreErrors when issuing a single-transaction batch call:

1. **Specify the ignoreErrors attribute when creating the batch object.** In this case, the attribute applies to all requests in the batch:

   ```perl
   my $batch = $N->newBatch('single', 'DuplicateResourceName');
   ```

2. **Specify the ignoreErrors attribute as an argument to an individual request.** In this case, the attribute applies only to that request and will override any global value specified:

   ```perl
   my $req2 = $batch->createResource($resource, {ignoreErrors => 'DuplicateResourceName'});
   ```

### Installing Commander Perl modules into Your Perl Distribution

You may want to use your existing Perl distribution. If so, Commander uses a CPAN style module, located in `<installdir>/src`, that can be installed with the following commands:

```
tar xzvf ElectricCommander-<your version>.tar.gz
cd ElectricCommander-<your version>
perl Makefile.PL
make install;# Use nmake on Windows
```
These commands install the Commander Perl and all of its submodules. If some prerequisite modules are missing, the `Makefile.PL` script will indicate which modules are needed.

## Installing Perl Modules into the Commander Perl Distribution

You may want expand the Commander Perl distribution by adding Perl modules from CPAN or third party vendors.

Install Perl modules using CPAN installer. The installer comes with the Commander Perl distribution in `<commanderDir>/perl/bin`.

### For Linux

From the command line use:

```
<commanderDir>/perl/bin/perl -MCPAN -e 'install <module>'
```

### For Windows

Compatibility with Commander is important. Commander 4.1 (and above) versions use Perl 5.8 for ec-perl.

If the Perl package is not Perl-only and requires compiling (for example, for C code):

- Use Windows Visual Studio VC6 (the same version used by Commander).
- Make sure that `cl` and `nmake` are both in your path. The Visual Studio install has a Command Prompt with these executables already in the path.

Extra steps are needed for Windows because of a problem with Perl and CPAN if you are running from a directory with spaces in the name. (By default, Commander has spaces in the installed directory.)

- Use a network drive to eliminate references to spaces.

  Use `subst` to mount the Perl directory under a different drive letter:

```
c:\> subst x: "c:\program files\electric cloud\electriccommander"
```

Start CPAN from the new location:

```
c:\> x:\perl\bin\perl -MCPAN -e shell
```

Configure CPAN to install into the new location:

```
cpan> o conf makepl_arg PREFIX=x:/perl
```

Install the module:

```
cpan> install <module>
```

Ending CPAN:

```
cpan> quit
```

- Change the `<commanderDir>/perl/lib/config.pm` file to eliminate spaces in references to the Commander path.
  
  For example:

```perl
#archlibexp = 'C:\Program Files\Electric Cloud\ElectricCommander\perl\lib',
archlibexp = 'X:\perl\lib',
#privlibexp = 'C:\Program Files\Electric Cloud\ElectricCommander\perl\lib',
privlibexp = 'X:\perl\lib',
#scriptdir = 'C:\Program Files\Electric Cloud\ElectricCommander\perl\lib',
scriptdir = 'X:\perl\lib',
#sitearchexp = 'C:\Program Files\Electric
```
Cloud\ElectricCommander\perl\site\lib',
sitearchexp => 'X:\perl\lib',
#sitelibexp => 'C:\Program Files\Electric Cloud\ElectricCommander\perl\site\lib',
sitelibexp => 'X:\perl\lib',

- Temporarily add X:\perl\bin to your Windows path.

**When Upgrading Commander**

During a Commander upgrade, the installer makes every attempt to preserve Perl packages. However, future Commander versions may contain an upgraded Perl version, which may then require a reinstall of any added Perl packages.
API commands - ACL Management

breakAclInheritance
checkAccess
createAclEntry
deleteAclEntry
getAccess
getAclEntry
modifyAclEntry
restoreAclInheritance

breakAclInheritance
Breaks ACL (access control list) inheritance at the given object. With inheritance broken, only the access control entries directly on the ACL will be considered.

You must specify locator arguments to find the object where you want to break inheritance.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactName</td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version.</td>
</tr>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms:</td>
</tr>
<tr>
<td></td>
<td>relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the</td>
</tr>
<tr>
<td></td>
<td>project that contains the request target object. A qualifying project name</td>
</tr>
<tr>
<td></td>
<td>is required.</td>
</tr>
<tr>
<td></td>
<td>absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the</td>
</tr>
<tr>
<td></td>
<td>credential can be from any specified project, regardless of the target</td>
</tr>
<tr>
<td></td>
<td>object's project.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the email configuration.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>The full name of the group. For Active Directory and LDAP, the full name if</td>
</tr>
<tr>
<td></td>
<td>the full DN.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by findObjects and getObjects. This value is a &quot;handle&quot; only for passing to API commands. The internal structure of this value is subject to change - do not parse this value.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure or a path to a procedure, including the name. <strong>Also requires</strong> projectName</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project - may be a path. The project name is ignored for credentials, procedures, steps, and schedules if they are specified as a path.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository used for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of a resource.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of a schedule - may be a path to a schedule. <strong>Also requires</strong> projectName</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step - may be a path to the step. <strong>Also requires</strong> projectName and procedureName</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>System objects names include: admin</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The full name of a user (for Active Directory or LDAP, this may be user@domain).</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of a workspace.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

#### Positional arguments

Arguments to locate the object, beginning with the top-level object locator.

#### Response

None or status OK message.

**ec-perl**

*Syntax:* $cmdr-&gt;breakAclInheritance({...});

*Example*

```perl
$cmdr-&gt;breakAclInheritance({ projectName =&gt; "Sample Project"});
```

**ectool**

*Syntax:* ectool breakAclInheritance ...

*Example*

```sh
ectool breakAclInheritance --projectName "Sample Project"
```

### checkAccess

Checks ACL (access control list) permission information associated with an object (including inherited ACLs) for the current user.

You must specify object locator arguments to define the object where you need to verify access.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
</tbody>
</table>
| credentialName   | The name of the credential container of the property sheet which owns the property. **credentialName** can be one of two forms:  
  - relative  
    (for example, "cred1") - the credential is assumed to be in the project that contains the request target object. Requires a qualifying project name.  
  - absolute  
    (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.                                                                 |
| environmentName  | The name of the environment container of the property sheet which owns the property; must be unique among all projects.                                                                                                                                                                                                                   |
| environmentTierName | The name of the environment tier container of the property sheet which owns the property.                                                                                                                                                                                                                                                  |
| gatewayName      | The name of the gateway container of the property sheet.                                                                                                                                                                                                                                                                                   |
| groupName        | The full name of the group container of the property sheet which owns the property.  
  For Active Directory and LDAP, this is a full DN.                                                                                                                                                                                                                                                                                  |
<p>| jobId            | The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.                                                                                                                                                                       |
| jobStepId        | The unique identifier (UUID) for a job step, assigned automatically when the job step is created.                                                                                                                                                                                                                                         |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by findObjects and getObjects. This value is a &quot;handle&quot; only for passing to API commands. The internal structure of this value is subject to change - do not parse this value.</td>
</tr>
<tr>
<td>path</td>
<td>Property path string.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin - the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure - may be a path to the procedure. Also requires projectName</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project container of the property sheet which owns the property that may be a path; must be unique among all projects. The project name is ignored for credentials, procedure, steps, and schedules if it is specified as a path.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule - may be a path to the schedule. Also requires projectName</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step - may be a path to the step. Also requires projectName and procedureName</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>System object names include: admin</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The full name of the user. For Active Directory and LDAP, the name may be user@domain.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

### Positional arguments
Arguments to locate the object, beginning with the top-level object locator.

### Response
For the specified object, returns the effective permissions for the current user.

**ec-perl**

```
syntax: $cmdr->checkAccess({...});
```

**Example**

```
$cmdr->checkAccess ({"projectName":"Sample Project"});
```

**ectool**

```
syntax: ectool checkAccess ...
```

**Example**

```
ectool checkAccess --projectName "Sample Project"
```

### createAclEntry
Creates an ACE (access control list entry) on an object for a given principal.

You must specify the `principalType,principalName,` and `locator` options for the object to modify.
## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
</tbody>
</table>
| credentialName    | **credentialName** can be one of two forms:  
  - **relative** (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.  
  - **absolute** (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object’s project. |
| configName        | The name of the email configuration.                                                                                                         |
| gatewayName       | The name of the gateway.                                                                                                                      |
| groupName         | The name of a group.                                                                                                                          |
| jobId             | The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template. |
| jobStepId         | The unique identifier for a job step, assigned automatically when the job step is created.                                                   |
| notifierName      | The name of the email notifier.                                                                                                               |
| objectId          | This is an object identifier returned by findObjects and getObjects.                                                                          |
| pluginName        | The name of the plugin - the plugin key for a promoted plugin or plugin key and version for an unpromoted plugin.                            |
| principalName     | This is either a user or a group name.                                                                                                        |
| principalType     | This is either user or group.                                                                                                                 |
| **Privileges:**    | <allow|deny>  
  If a privilege is not specified, permission is set to inherit from its parent object's ACL.                                                |
<p>| procedureName     | The name of the procedure. <strong>Also requires</strong> projectName                                                                                      |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step.</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>System object names include: admin</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The full name of the user.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

**Positional arguments**

```
principalType,principalName
```

**Response**

None or status OK message.
**API commands - ACL Management**

**ec-perl**

*Syntax:* `cmdr->createAclEntry(<principalType> <principalName>, {...});`

*Example*

```
cmdr->createAclEntry("user", "j smith", {"projectName":"Sample Project", "readPrivilege":"allow", "modifyPrivilege":"deny", "executePrivilege":"deny", "changePermissionsPrivilege":"deny"});
```

**ectool**

*Syntax:* `ectool createAclEntry <principalType> <principalName> ...

*Example*

```
ectool createAclEntry user "j smith" --projectName "Sample Project" --readPrivilege allow
   --modifyPrivilege deny --executePrivilege deny --changePermissionsPrivilege deny
```

**deleteAclEntry**

Deletes an ACE (access control list entry) on an object for a given principal.

You must specify a `principalType` and `principalName` and you must use locator arguments to specify the location for this ACL entry.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>artifactName</strong></td>
<td>The name of the artifact.</td>
</tr>
</tbody>
</table>
| **artifactVersionName**| The name of the artifact version.  
  **Note:** An artifact version name is interpreted by the server as the `artifactVersionName` attribute for the `artifactVersion` in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name—the Commander server interprets either name form correctly. |
| **credentialName**| `credentialName` can be one of two forms:  
  **relative**  
  (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.  
  **absolute**  
  (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project. |
<p>| <strong>configName</strong>   | The name of the email configuration.                                         |
| <strong>gatewayName</strong>  | The name of the gateway.                                                    |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName</td>
<td>The name of a group whose ACL entry you want to delete.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>objectId</td>
<td>An object identifier returned by findObjects and getObjects.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier whose ACL entry you want to delete.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin whose ACL entry you want to delete.</td>
</tr>
<tr>
<td>principalName</td>
<td>This is either the user or the group name.</td>
</tr>
<tr>
<td>principalType</td>
<td>This is either a user or a group &lt;user</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure whose ACL entry you want to delete. <strong>Also requires</strong> projectName, where this procedure is a member.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project where you are deleting an ACL entry.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource whose ACL entry you want to delete.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule whose ACL entry you want to delete. <strong>Also requires</strong> projectName from which this schedule runs procedures.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step whose ACL entry you want to delete. <strong>Also requires</strong> projectName and procedureName to indicate where this step resides.</td>
</tr>
</tbody>
</table>
### Arguments | Descriptions
--- | ---

**systemObjectName** | System object names include: 
admin|directory|licensing|log|plugins|priority| 
projects|resources|server|session|workspaces

**transitionDefinitionName** | The name of the transition definition.

**transitionName** | The name of the transition.

**userName** | The name of the user who's ACL entry you want to delete.

**workflowDefinitionName** | The name of the workflow definition.

**workflowName** | The name of the workflow.

**workspaceName** | The name of the workspace whose ACL entry you want to delete.

**zoneName** | The name of the zone.

#### Positional arguments

*principalType*, *principalName*

#### Response

None or a status OK message.

#### ec-perl

*Syntax:* `$cmdr->deleteAclEntry(<principalType>, <principalName>, {<optionals>});`

*Example*

```
$cmdr->deleteAclEntry('user', 'j smith', {projectName => 'Sample Project'});
```

#### ectool

*Syntax:* `ectool deleteAclEntry <principalType> <principalName> ...

*Example*

```
ectool deleteAclEntry user "j smith" --projectName "Sample Project"
```

### getAccess

Retrieves ACL information (access control list) associated with an object, including inherited ACLs.

You must specify object locators to find the object where you need to verify access.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the credential container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential container of the property sheet which owns the property. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
</tbody>
</table>
| emulateRestoreInheritance  | Whether or not to include one level of broken inheritance if it exists. Used for seeing what access would look like if the lowest level of broken inheritance was restored.  
<Boolean flag - 0|1|true|false> If set to 1, this argument returns ACL information to what it would be if inheritance were restored on this object. |
<p>| environmentName            | The name of the environment container of the property sheet which owns the property; must be unique among all projects.                      |
| environmentTierName        | The name of the environment tier container of the property sheet which owns the property.                                                   |
| gatewayName                | The name of the gateway container of the property sheet.                                                                                     |
| groupName                  | The name of the group container of the property sheet that owns the property.                                                                |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier that contains the ACL.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by findObjects and getObjects.</td>
</tr>
<tr>
<td>path</td>
<td>Property path string.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin that contains the ACL.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the ACL. Also requires projectName</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains the ACL; must be unique among all projects.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource that contains the ACL.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing the ACL. Also requires projectName</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step containing the ACL. Also requires projectName and procedureName</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
`systemObjectName` | System objects include: admin|artifactVersions|directory|emailConfigs|log|plugins|server|session|workspaces
`transitionDefinitionName` | The name of the transition definition.
`transitionName` | The name of the transition.
`userName` | The name of the user that contains the ACL.
`workflowDefinitionName` | The name of the workflow definition.
`workflowName` | The name of the workflow.
`workspaceName` | The name of the workspace that contains the ACL.
`zoneName` | The name of the zone.

**Positional arguments**
Arguments to specify the object, beginning with the top-level object locator.

**Response**
One or more `object` elements, each consisting of one or more `aclEntry` elements. Each `object` represents an object in the ACL inheritance chain starting with the most specific object. Each `aclEntry` identifies a user or group and the privileges granted or denied by the entry, and includes a `breakInheritance` element if applicable.

**ec-perl**

```perl
syntax:  $cmdr-&gt;getAccess({<optionals>});

Example
$cmdr-&gt;getAccess({projectName =&gt; "Sample Project"});
```

**ectool**

```bash
syntax:  ectool getAccess ...

Example
ectool getAccess --projectName "Sample Project"
```

**Back to Top**

**getAcclEntry**

Retrieves an ACE (access control entry list) on an object for a given principal.
You must specify a `principalType`, `principalName`, and an object locator to specify which ACE to examine.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>artifactName</code></td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td><code>artifactVersionName</code></td>
<td>The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td><code>credentialName</code></td>
<td><code>credentialName</code> can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td><code>configName</code></td>
<td>The name of the email configuration.</td>
</tr>
<tr>
<td><code>gatewayName</code></td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td><code>groupName</code></td>
<td>The name of the group.</td>
</tr>
<tr>
<td><code>jobId</code></td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td><code>jobStepId</code></td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td><code>notifierName</code></td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td><code>objectId</code></td>
<td>This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</td>
</tr>
<tr>
<td><code>pluginName</code></td>
<td>The name of the plugin - the plugin key for a promoted plugin or plugin key and version for an unpromoted plugin.</td>
</tr>
<tr>
<td><code>principalName</code></td>
<td>This is either the user or group name.</td>
</tr>
<tr>
<td><code>principalType</code></td>
<td>This is either user or group.</td>
</tr>
<tr>
<td><code>procedureName</code></td>
<td>The name of the procedure. Also requires <code>projectName</code></td>
</tr>
<tr>
<td><code>projectName</code></td>
<td>The name of the project.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
</tbody>
</table>
| scheduleName       | The name of the schedule.  
  **Also requires** projectName |
| stateDefinitionName| The name of the state definition.                                            |
| stateName          | The name of the state.                                                       |
| stepName           | The name of the step.  
  **Also requires** projectName and procedureName |
| systemObjectName   | System objects include:  
  admin|artifactVersions|directory|emailConfigs|log|plugins|server|session|workspaces |
| transitionDefinitionName | The name of the transition definition.                                      |
| transitionName     | The name of the transition.                                                  |
| userName           | The full name of the user.                                                   |
| workflowDefinitionName | The name of the workflow definition.                                        |
| workflowName       | The name of the workflow.                                                    |
| workspaceName      | The name of the workspace.                                                   |
| zoneName           | The name of the zone.                                                        |

**Positional arguments**  
principalType, principalName

**Response**  
One aclEntry element.

**ec-perl**  
**syntax:** $cmdr->getAclEntry(<principalType>, < principalName>, {...});
**Example**

```bash
$cmdr->getAclEntry("user", "j smith", \{projectName => "Sample Project"\});
```

**ectool**

**syntax:** ectool getAclEntry <principalType> <principalName> ...

**Example**

```bash
ectool getAclEntry --principalType "user" --principalName "j smith" --projectName "Sample Project"
```

**modifyAclEntry**

Modifies an ACE (access control list entry) on an object for a given principal.

**Note:** If a privilege is not specified, it inherits from its parent object’s ACL.

You must specify `principalType`, `principalName` and object locator arguments to identify the target ACL.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactName</td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>credentialName</td>
<td><code>credentialName</code> can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the email configuration.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group containing the ACL entry.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier containing the ACL entry.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by findObjects and getObjects.</td>
</tr>
<tr>
<td>plugFileName</td>
<td>The name of the plugin containing the ACL entry.</td>
</tr>
<tr>
<td><strong>Privileges:</strong></td>
<td></td>
</tr>
<tr>
<td>readPrivilege</td>
<td></td>
</tr>
<tr>
<td>modifyPrivilege</td>
<td></td>
</tr>
<tr>
<td>executePrivilege</td>
<td></td>
</tr>
<tr>
<td>changePermissionsPrivilege</td>
<td></td>
</tr>
<tr>
<td>&lt;allow</td>
<td>deny&gt;</td>
</tr>
<tr>
<td>principalName</td>
<td>This is either the user or group name.</td>
</tr>
<tr>
<td>principalType</td>
<td>This is either user or group.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the ACL entry.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the ACL entry.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource containing the ACL entry.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing the ACL entry.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step containing the ACL entry.</td>
</tr>
<tr>
<td><strong>Also requires</strong> projectName</td>
<td></td>
</tr>
<tr>
<td>systemObjectName</td>
<td>System object names include: admin</td>
</tr>
</tbody>
</table>
### Arguments and Descriptions

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user containing the ACL entry.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace containing the ACL entry.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

#### Positional arguments

- principalType, principalName

#### Response

None or a status OK message.

### ec-perl Syntax

**Syntax:**

```
$cmdr->modifyAclEntry(<principalType>, <principalName>, {<optionals>});
```

**Example**

```
$cmdr->modifyAclEntry("user", "j smith", {projectName => "Sample Project", modifyPrivilege => "deny", });
```

### ectool Syntax

**Syntax:**

```
ectool modifyAclEntry <principalType> <principalName> ...
```

**Example**

```
ectool modifyAclEntry user "j smith" --projectName "Sample Project" --modifyPrivilege deny
```

### restoreAclInheritance

Restores ACL (access control list) inheritance for the specified object.

**Note:** You must use object locators to specify the object where you want to restore ACL inheritance.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| artifactVersionName | The name of the artifact version.  
**Note:** An artifact version name is interpreted by the server as the **artifactVersionName** attribute for the **artifactVersion** in question.  
This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name—the Commander server interprets either name form correctly. |
| credentialName     | The name of the credential whose ACL inheritance you want to restore.  
**credentialName** can be one of two forms:  
**relative**  
for example, "cred1" - the credential is assumed to be in the project that contains the request target object.  
**absolute**  
(for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.  
**Also requires** **projectName** |
| configName         | The name of the email configuration.                                                                                                                                                                  |
| gatewayName        | The name of the gateway.                                                                                                                                                                             |
| groupName          | The name of the group whose ACL inheritance you want to restore.                                                                                                                                     |
| jobId              | The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.                           |
| jobStepId          | The unique identifier for a job step, assigned automatically when the job step is created.                                                                                                           |
| notifierName       | The name of the email notifier whose ACL inheritance you want to restore.  
**Also requires** **projectName** and **procedureName**;  
**projectName**, **procedureName**, and **stepName**; **jobId** or **jobStepId** |
| objectId           | This is an object identifier returned by **findObjects** and **getObjects**.                                                                                                                         |
| pluginName         | The name of the plugin whose ACL inheritance you want to restore.                                                                                                                                   |
| procedureName      | The name of the procedure whose ACL inheritance you want to restore.  
**Also requires** **projectName** |
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project whose ACL inheritance you want to restore.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource whose ACL inheritance you want to restore.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule whose ACL inheritance you want to restore.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step whose ACL inheritance you want to restore.</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>The name of the system object whose ACL inheritance you want to restore.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user whose ACL inheritance you want to restore.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace whose ACL inheritance you want to restore.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

### Positional arguments

Arguments to locate the object, beginning with the top-level object locator.
Response
   None or a status OK message.

ec-perl
   syntax: $cmdr->restoreAclInheritance({<optionals>});

   Example
   $cmdr->restoreAclInheritance({projectName => "Sample Project"});

ectool
   syntax: ectool restoreAclInheritance ...

   Example
   ectool restoreAclInheritance --projectName "Sample Project"

Back to Top
createApplication

Creates a new application for a project.

You must specify the projectName and the applicationName arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricCommander.</td>
</tr>
</tbody>
</table>

Response

Returns an application element.

**ec-perl**

* syntax: $<object>->createApplication(<projectName>, <applicationName>, {<optionals>});

* Example

  $ec->createApplication("Default", "app1", {description => "aDescription"});

**ectool**

* syntax: ectool createApplication <projectName> <applicationName> [optionals...]

* Example

  ectool createApplication default newApp --description aDescription

Back to Top
**deleteApplication**

Delete an application.

You must specify the **projectName** and the **applicationName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

None or a status OK message.

**ec-perl**

*syntax:* `$<object>-deleteApplication (<projectName>, <applicationName>);

*Example*

```
$ec->deleteApplication ("Default", "appToDelete");
```

**ectool**

*syntax:* `ectool deleteApplication <projectName> <applicationName>

*Example*

```
ectool deleteApplication default appToDelete
```

**getApplication**

Finds an application by name.

You must specify the **projectName** and the **applicationName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>
Response
Retrieves the specified application element.

**ec-perl**
*syntax:* \$<object>-getApplication(<projectName>, <applicationName>);

*Example*
\$ec->getApplication("Default", "newApp");

**ectool**
*syntax:* ectool getApplication <projectName> <applicationName>

*Example*
ectool getApplication default newApp

**getApplications**
Retrieves all applications in a project.
You must specify the projectName argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>Argument Type: String</td>
<td></td>
</tr>
</tbody>
</table>

Response
Retrieves zero or more application elements.

**ec-perl**
*syntax:* \$<object>-getApplications(<projectName>);

*Example*
\$ec->getApplications("Default");

**ectool**
*syntax:* ectool getApplications <projectName>

*Example*
ectool getApplications default

Back to Top
modifyApplication

Modifies an existing application.

You must specify the `projectName` and the `applicationName` arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application; must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricCommander. Argument Type: String</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed. Argument Type: String</td>
</tr>
</tbody>
</table>

Response

Retrieves an updated application element.

**ec-perl**

 syntax: ```<object>-modifyApplication(<projectName>, <applicationName>, {<optionals>}); ```

Example

```ec>-modifyApplication("Default", "app1", {newName=> "newAppName", description => "exampleText"); ```

**ectool**

 syntax: `ectool modifyApplication <projectName> <applicationName> [optionals...]` 

Example

`ectool modifyApplication default newApp --newName modApp --description exampleText`
API commands - Application Tier

createApplicationTier

deleteApplicationTier

getApplicationTier

getApplicationTiersInComponent

modifyApplicationTier

createApplicationTier

Creates a new application tier in the application.

You must specify the projectName, applicationName, and applicationTierName arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>Name of the tier; must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricCommander.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Response

Returns an application tier element.

ec-perl

**syntax:**

```perl
$<object>->createApplicationTier(<projectName>, <applicationName>, <applicationTierName>, {<optionals>});
```

**Example**

```perl
$ec->createApplicationTier("Default", "app1", "appTier2", {description=&gt; "example_text"});
```

ectool

**syntax:**
ectool createApplicationTier <projectName> <applicationName> <applicationTierName> [optionals...]
**deleteApplicationTier**

Deletes a tier from an application.

You must specify the `projectName`, `applicationName`, and `applicationTierName` arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td><code>applicationTierName</code></td>
<td>Name of the tier; must be unique within the application.</td>
</tr>
</tbody>
</table>

**Response**

None or a status OK message.

**ec-perl**

*Syntax:*
```
$<object>->deleteApplicationTier(<projectName>, <applicationName>, <applicationTierName>);
```

*Example*
```
$ec->deleteApplicationTier("Default", "app1", "appTierToDelete");
```

**ectool**

*Syntax:* `ectool deleteApplicationTier <projectName> <applicationName> <applicationTierName>`

*Example*
```
ectool deleteApplicationTier default newApp appTierToDelete
```

**getApplicationTier**

Finds an application tier by name.

You must specify the `projectName`, `applicationName`, and `applicationTierName` arguments.
**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>Name of the tier; must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

Retrieves an application tier element.

**ec-perl**

*syntax:* `$<object>-getApplicationTier(<projectName>, <applicationName>, <applicationTierName>);`

*Example*

```perl
$ec->getApplicationTier("Default", "app1", "appTier2");
```

**ectool**

*syntax:* `ectool getApplicationTier <projectName> <applicationName> <applicationTierName>

*Example*

`ectool getApplicationTier default newApp appTier1`

Back to Top

---

**getApplicationTiers**

Retrieves all application tiers in an application.

You must specify the `projectName` and `applicationName` arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

Retrieves zero or more application tier elements.
### getApplicationTiers

**Syntax:** $<object>-getApplicationTiers(<projectName>, <applicationName>);

**Example**

$ec->getApplicationTiers("Default", "appl");

**ectool**

**Syntax:** ectool getApplicationTiers <projectName> <applicationName>

**Example**

ectool getApplicationTiers default newApp

### getApplicationTiersInComponent

Retrieves all application tiers that are used by the given component.

You must specify the **projectName** and the **componentName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projectName</strong></td>
<td>Name for the project; must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td><strong>componentName</strong></td>
<td>Name of the component. Argument Type: String</td>
</tr>
<tr>
<td><strong>applicationName</strong></td>
<td>(Optional) Name of an application to which this component is scoped. Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

Retrieves zero or more application tier elements used by the specified component.

**ec-perl**

**Syntax:** $<object>-getApplicationTiersInComponent(<projectName>, <componentName>, {<optionals>});

**Example**

$ec->getApplicationTiersInComponent("default", "newComponent");

**ectool**

**Syntax:** ectool getApplicationTiersInComponent <projectName> <componentName> [optionals...]

---

Back to Top
modifyApplicationTier

Modifies an existing tier in the application.

You must specify the **projectName**, **applicationName**, and **applicationTierName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>Name of the tier; must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricCommander.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Response

Retrieves an updated application tier element.

**ec-perl**

```perl
$<object>-modifyApplicationTier(<projectName>, <applicationName>, <applicationTierName>, {<optionals>});
```

**Example**

```
$ec->modifyApplicationTier("Default", "app1", "appTier2",
   {newName=> "appTierB", description=> "newText"});
```

**ectool**

```ectool modifyApplicationTier <projectName> <applicationName>
     <applicationTierName> [optionals...]```
Example

ectool modifyApplicationTier default newApp appTier1
--description new_exampleText --newName appTierA

Back to Top
addDependentsToArtifactVersion

Adds an artifact version query to an existing artifact. Dependent artifact versions are retrieved when the parent artifact version is retrieved.

You must specify an artifactVersionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. Note: An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>dependentArtifactVersions</td>
<td>One or more artifact version queries. The most current match of each query is retrieved when the primary artifact is retrieved. Dependent artifact version query strings are in this form: &lt;groupId&gt;:&lt;artifactKey&gt;:&lt;versionRange&gt; (versionRange is optional). The version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.</td>
</tr>
</tbody>
</table>

Positional arguments

artifactVersionName

Response

None or status OK message.

getArtifactVersions
**ec-perl**

*Syntax:* `$cmdr->addDependentsToArtifactVersion (<artifactVersionName>,
{<optionals>});`

*Example*

# Add a dependency on cmdr:SDK:1.2.0 and the most current version of core:infra that
# is greater than or equal to 2.1.0.
$cmdr->addDependentsToArtifactVersion ({artifactVersionName => "myGroup:myAKey:1.0.0-55",
dependentArtifactVersions => ["cmdr:SDK:1.2.0", "core:infra:[2.1.0,]"]});

**ectltool**

*Syntax:* `ectool addDependentsToArtifactVersion <artifactVersionName>...`

*Example*

`ectool addDependentsToArtifactVersion --artifactVersionName "myGroup:myAKey:1.0.0-55",
--dependentArtifactVersions "cmdr:SDK:1.2.0" "core:infra:[2.1.0,]"

**cleanupArtifactCache**

Deletes stale artifact versions from an artifact cache. A "stale artifact version" is one whose metadata was previously deleted from the Commander server.

*Note:* If you are not logged in as "admin", you cannot use this command. However, using the `force` option overrides admin login privileges.

You must specify a `cacheDirectory`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>cacheDirectory</td>
<td>The directory where stale artifact versions are stored.</td>
</tr>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `cacheDirectory`

**Response**

- Returns a list of directories that were deleted.

**ec-perl**

*Syntax:* `$cmdr->cleanupArtifactCache(<cacheDirectory>);`


**Example**

```bash
$cmdr->cleanupArtifactCache("/var/artifact-cache");
```

**ectool**

**syntax:** `ectool cleanupArtifactCache <cacheDirectory>`

**Example**

```bash
ectool cleanupArtifactCache "/var/artifact-cache"
```

---

**cleanupRepository**

Deletes stale artifact versions from the repository backing-store. A "stale artifact version" is one whose metadata was previously deleted from the Commander server.

**Note:** If you are not logged in as "admin", you cannot use this command. However, using the `force` option overrides admin login privileges.

You must specify a `backingStoreDirectory`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>backingStoreDirectory</td>
<td>The repository directory where artifact versions are stored.</td>
</tr>
</tbody>
</table>

<Boolean flag - 0|1|true|false> If set to "true", this option can be used so you can cleanup the repository even if the g/a/v's in the directory specified do not match up with any artifacts reported by the server. By default, this is false, and helps users avoid deleting arbitrary directory trees if they did not specify the repository backingstore properly.

**Positional arguments**

`backingStoreDirectory`

**Response**

Returns a list of directories that were deleted.

**ec-perl**

**syntax:** `$cmdr->cleanupRepository(<backingStoreDirectory>);`

**Example**

```perl
use strict;
use ElectricCommander;
my $cmdr = ElectricCommander->new({debug => 1});
$cmdr->login("admin", "changeme");
$cmdr->cleanupRepository("/var/repository-data");
```
**ectool**

*Syntax:* `ectool cleanupRepository <backingStoreDirectory>`

*Example*

```bash
ectool cleanupRepository "/var/repository-data"
```

[Back to Top](#)

---

**createArtifact**

Creates a new artifact.

You must specify a `groupId` and an `artifactKey`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>artifactKey</code></td>
<td>User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td><code>artifactVersionNameTemplate</code></td>
<td>A template for the names of artifact versions published to this artifact. This option overrides the value set in the server settings for &quot;artifact name template.&quot;. The global setting can be manipulated in the Server Settings page (Administration &gt; Server, select the Settings link).</td>
</tr>
<tr>
<td><code>description</code></td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td><code>groupId</code></td>
<td>A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, spaces, underscores, hyphens, and periods.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`groupId`, `artifactKey`

**Response**

Returns an `artifact` element.

**ec-perl**

*Syntax:* `$cmdr->createArtifact(<groupId>, <artifactKey>, {<optionals>});`

*Example*

```perl
$cmdr->createArtifact("thirdPartyTools", "SDK", {description => "3rd party tools SD K"});
```
ectool

**syntax:**  ectool createArtifact <groupId> <artifactKey> ...

**Example**

ectool createArtifact thirdPartyTools SDK --description "3rd party tools SDK"

---

**createRepository**

Creates a repository for one or more artifacts.

You must specify a repositoryName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>repositoryDisabled</td>
<td>`&lt;Boolean flag -0</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository.</td>
</tr>
<tr>
<td>url</td>
<td>The URL to use to communicate with the repository server.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone where this repository resides.</td>
</tr>
</tbody>
</table>

**Positional arguments**

repositoryName

**Response**

Returns a repository element.

**ec-perl**

**syntax:**  $cmdr-&gt;createRepository(<repositoryName>, [<optionals>]);

**Example**

$cmdr-&gt;createRepository("myRepos", {repositoryDisabled =&gt; "true", url =&gt; "https://test.ecloud.com:8200"});

**ectool**

**syntax:**  ectool createRepository <repositoryName> ...
**Example**

ectool createRepository myRepos --repositoryDisabled "true" --url "https://test.ecloud.com:8200"

deleteArtifact

Deletes an existing artifact element and all artifact versions.

You must specify an `artifactName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>artifactName</code></td>
<td>The name of the artifact to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`artifactName`

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* 
```perl
$cmdr->deleteArtifact(<artifactName>);
```

*Example*

```perl
$cmdr->deleteArtifact("commander:SDK");
```

**ectool**

*Syntax:* 
```bash
ectool deleteArtifact <artifactName>
```

*Example*

```bash
ectool deleteArtifact "commander:SDK"
```

deleteArtifactVersion

Deletes artifact version metadata from the Commander database.

(This API call does not delete or remove artifacts stored on the repository machine.)

You must specify an `artifactVersionName`. 
API commands - Artifact Management

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. Note: An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
</tbody>
</table>

Positional arguments
artifactVersionName

Response
None or a status OK message.

ec-perl

*Syntax:* $cmdr->deleteArtifactVersion(<artifactVersionName>);

*Example*
$cmdr->deleteArtifactVersion("myGroup:myKey:1.0.0-55");

ectool

*Syntax:* ectool deleteArtifactVersion <artifactVersionName>

*Example*

ectool deleteArtifactVersion "myGroup:myKey:1.00.0-55"

Back to Top

deleteRepository

Deletes artifact repository metadata from the Commander database. (This API call does not delete or remove artifacts stored on the repository machine.)

You must supply a repositoryName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository to delete.</td>
</tr>
</tbody>
</table>

Positional arguments
repositoryName

Response
None or a status OK message.
**findArtifactVersions**

This command returns the most current artifact version that matches the filter criteria and its dependent artifact versions.

This API implicitly searches for artifact versions in the "available" state, and if run in a job step, registers the step as a retriever for the returned artifact versions.

Because of the complexity of specifying filter criteria, this API is not supported by *ectool*. However, all of its capabilities are supported through the Perl API.

**Note:** The *retrieveArtifactVersions* API uses this API to find the appropriate artifact version in the Commander server and then retrieves the artifact version from a repository. You may prefer to use the *retrieveArtifactVersions* API instead of this API because while this API returns slightly different information, it also has the side-effect of "retriever step registration" mentioned above.

You must specify an artifactName or a groupId with an artifactKey.
Arguments | Descriptions
--- | ---
filter | A list of zero or more filter criteria definitions used to define objects to find.
  Each element of the filter list is a hash reference containing one filter criterion. You may specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.

Two types of filters:
  "property filters" - used to select objects based on the value of the object's intrinsic or custom property
  "boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic.

Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property.

Property filter operators are:
  between (2 operands)
  contains (1)
  equals (1)
  greaterOrEqual (1)
  greaterThan (1)
  in (1)
  lessOrEqual (1)
  lessThan (1)
  like (1)
  notEqual (1)
  notLike (1)
  isNotNull (0)
  isNull (0)

A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter.

Boolean operators are:
  not (1 operand)
  and (2 or more operands)
  or (2 or more operands)

artifactKey | User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
artifactName | The name of an artifact.
artifactVersionName | The name of an artifact version.
Arguments | Descriptions
--- | ---
groupId | A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.

includeDependents | Options are:
- 0/false – dependent artifacts are not retrieved.
- 1/true – dependent artifacts are retrieved.

jobStepId | The unique identifier for the job step (if any), that is making the request. This job step will be marked as a retriever for the matching artifact versions.

versionRange | The range of versions to search. Version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.

### Positional arguments
None

### Response
This command returns zero or more artifactVersion elements. In addition, this API returns a searchDetails element with text describing how the server evaluated candidate artifact versions and ultimately decided to return the result artifactVersion and its dependent(s).

**ec-perl**

**syntax:** $cmdr->findArtifactVersions({<optionals>});

**Example 1**

# Find the most current core:infra artifact version whose version is 1.x.x.
$cmdr->findArtifactVersions({groupId => "core",
    artifactKey => "infra",
    versionRange => "[1.0, 2.0)"});

# Or alternatively ...
$cmdr->findArtifactVersions({artifactName => "core:infra",
    versionRange => "[1.0,2.0)"});

**Example 2**

# Find the most current core:infra artifact version with QA approval level 3 or above.
$cmdr->findArtifactVersions({groupId => "core",
    artifactKey => "infra",
    filter => {propertyName => "qaLevel",}});
getArtifact

Retrieves an artifact by name.

You must specify an artifactName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactName</td>
<td>The name of the artifact.</td>
</tr>
</tbody>
</table>

**Positional arguments**

artifactName

**Response**

Retrieves an artifact element.

**ec-perl**

*Syntax:* `$cmdr->getArtifact (<artifactName>);

*Example*

`$cmdr-> getArtifact("myGroup:myKey");`

**ectool**

*Syntax:* `ectool getArtifact <artifactName>

*Example*

`ectool getArtifact "myGroup:myKey"`

getArtifacts

Retrieves all artifacts in the system.

You must specify search filter criteria to find the artifacts you need.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
**Positional arguments**

None

**Response**

Zero or more `artifact` elements.

**ec-perl**

*Syntax:* \$cmdr->getArtifacts();

*Example*

\$cmdr->getArtifacts();

**ectool**

*Syntax:* ectool getArtifacts

*Example*

ectool getArtifacts

---

**getArtifactVersion**

Retrieves an artifact version by its name.

You must specify an `artifactVersionName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version to retrieve. <strong>Note:</strong> An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>includeRetrieverJobs</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>includeRetrieverJobSteps</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>maxRetrieverJobs</td>
<td>If one of the includeRetriever* options are specified, return at most &quot;this many&quot; of the most recent retrievers. Without this option, the Commander server will return all retrievers.</td>
</tr>
</tbody>
</table>
**Positional arguments**

artifactVersionName

**Response**

One artifactVersion element. If includeRetrieverJobs or includeRetrieverJobSteps is set, the artifactVersion element will contain zero or more retriever child elements, each containing retriever information for one job or job step.

**ec-perl**

* syntax: $cmdr->getArtifactVersion(<artifactVersionName>, {<optionals>});

* Example

```perl
$cmdr->getArtifactVersion("myGroup:myKey:1.0.0-55", {includeRetrieverJobs => "true"});
```

**ectool**

* syntax: ectool getArtifactVersion <artifactVersionName> ...

* Example

```bash
ectool getArtifactVersion myGroup:myKey:1.0.0-55 --includeRetrieverJobs "true"
```

**getArtifactVersions**

Retrieves all artifact versions in the system, filtered by artifact name, retriever job ID, and/or retriever job step ID.

You must specify search filter criteria to find the artifact versions you need.

If you do not provide any options, all artifact versions in the system are returned.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactName</td>
<td>The name of the artifact for the versions to retrieve.</td>
</tr>
<tr>
<td>retrieverJobId</td>
<td>The job ID that retrieved an artifact.</td>
</tr>
<tr>
<td>retrieverJobStepId</td>
<td>The job step ID that retrieved an artifact.</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Zero or more artifactVersion elements.

**ec-perl**

* syntax: $cmdr->getArtifactVersions({<optionals>});
Example

$cmdr->getArtifactVersions({artifactName => "myGroup:myKey"});

ectool

**syntax:** ectool getArtifactVersions ...

**Example**

ectool getArtifactVersions --artifactName "myGroup:myKey"

Back to Top

getManifest

Retrieves the manifest for a specified artifact version. The manifest includes a list of files and directories in the artifact version and its checksum file.

You must specify the **artifactVersionName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version whose manifest you want to retrieve.</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Manifest information for the specified artifact version: returns an XML stream containing any number of file elements, including the file name, file size, and "sha1" hashes for every file in the **artifactVersionName**.

ec-perl

**syntax:** $cmdr->getManifest(<artifactVersionName>);

**Example**

my ($manifest,$diagnostics) = $cmdr->getManifest("myGroup:myKey:1.0.0-55");

ectool

**syntax:** ectool getManifest <artifactVersionName>

**Example**

ectool getManifest myGroup:myKey:1.0.0-55

getRepositories

Retrieves all artifact repository objects known to the Commander server.
### getRepositories

Retrieves an artifact repository by its name.

You must specify a `repositoryName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository to retrieve.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `repositoryName`

**Response**

- Zero or more `repository` elements.

#### ec-perl

- **syntax:** `$cmdr->getRepositories ();`

  **Example**

  ```
  $cmdr->getRepositories ();
  ```

#### ectool

- **syntax:** `ectool getRepositories`

  **Example**

  ```
  ectool getRepositories
  ```

---

### getRepository

Retrieves an artifact repository by its name.

You must specify a `repositoryName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository to retrieve.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `repositoryName`

**Response**

- One `repository` element.

#### ec-perl

- **syntax:** `$cmdr->getRepository(<repositoryName>);`

  **Example**

  ```
  $cmdr->getRepository("myRepository");
  ```
modifyArtifact

Modifies an existing artifact.

You must specify an artifactName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactName</td>
<td>The name of the artifact to modify.</td>
</tr>
<tr>
<td>artifactVersionNameTemplate</td>
<td>A template for the names of artifact versions published to this artifact.</td>
</tr>
<tr>
<td></td>
<td>This option overrides the value set in the server settings for artifact names.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object.</td>
</tr>
<tr>
<td></td>
<td>If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags.</td>
</tr>
<tr>
<td></td>
<td>The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
</tbody>
</table>

Positional arguments

artifactName

Response

None or a status OK message.

ec-perl

syntax: $cmdr->modifyArtifact(<artifactName>, [<optionals>]);

Example

$cmdr->modifyArtifact("thirdParty-SDK", {description => "contains artifact versions for SDK"]);
**Example**

ectool modifyArtifact thirdParty-SDK --description "contains artifact versions for SDK"

modifyArtifactVersion

Modifies an existing artifact version.

You must specify an artifactVersionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version to modify.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>artifactVersionState</td>
<td>The state of the artifact version.</td>
</tr>
<tr>
<td></td>
<td>`&lt;publishing</td>
</tr>
<tr>
<td>dependentArtifactVersions</td>
<td>One or more artifact version queries. The most current match for each query is retrieved when the primary artifact is retrieved. Dependent artifact version query strings are in this form: <code>&lt;groupId&gt;:&lt;artifactKey&gt;:&lt;versionRange&gt;</code> (version range is optional).  <strong>Note:</strong> The absence of this argument does not clear or modify the dependent artifact version list for this artifact version.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; </code>&lt;b&gt; <code>&lt;br&gt; </code>&lt;div&gt; <code>&lt;dl&gt; </code>&lt;font&gt; <code>&lt;i&gt; </code>&lt;li&gt; <code>&lt;ol&gt; </code>&lt;p&gt; <code>&lt;pre&gt;</code> <code>&lt;span&gt;</code> <code>&lt;style&gt;</code> <code>&lt;table&gt;</code> <code>&lt;tc&gt;</code> <code>&lt;td&gt;</code> <code>&lt;th&gt;</code> <code>&lt;tr&gt;</code> <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>newName</td>
<td>Any name you choose to use as the new name for this artifact version.</td>
</tr>
<tr>
<td>removeAllDependentArtifactVers</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository.</td>
</tr>
</tbody>
</table>

**Positional arguments**

artifactVersionName
Response
None or a status OK message.

ec-perl

.syntax: $cmdr->modifyArtifactVersion('<artifactVersionName>', <optionals>);

Example
$cmdr->modifyArtifactVersion("myGroup:myKey:1.0.1-42375", {artifactVersionState => "unavailable"});

ectool

.syntax: ectool modifyArtifactVersion <artifactVersionName> ...

Example
ectool modifyArtifactVersion "myGroup:myKey:1.0.1-57385" --artifactVersionState unavailable

modifyRepository

Modifies an existing artifact repository.
You must specify a repositoryName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must</td>
</tr>
<tr>
<td></td>
<td>surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed</td>
</tr>
<tr>
<td></td>
<td>in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the repository.</td>
</tr>
<tr>
<td>repositoryDisabled</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>this option, the state of the repository is unchanged.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository.</td>
</tr>
<tr>
<td>url</td>
<td>The URL used to communicate with the artifact repository.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone where this repository resides.</td>
</tr>
</tbody>
</table>

Positional arguments

repositoryName
Response
Returns a modified repository element.

dc-perl

**syntax:** $cmdr->modifyRepository(<repositoryName>, [<optionals>]);

**Example**
$cmdr->modifyRepository("myNewRepos", {newName => "cmdrRepository"]);

dc-perl

**syntax:** $cmdr->modifyRepository(<repositoryName>, [<optionals>]);

**Example**
$cmdr->modifyRepository("myNewRepos", {newName => "cmdrRepository"]);

**Response**
Returns a modified repository element or an error if the repository does not exist.

dc-perl

**syntax:** $cmdr->moveRepository(<repositoryName>, [<optionals>]);

**Example**
$cmdr->moveRepository("reposThree", {beforeRepositoryName => "reposOne"]);

**ectool**

**syntax:** ectool moveRepository <repositoryName> ...

**Example**
ectool moveRepository myNewRepos --newName cmdrRepository

**moveRepository**

Moves an artifact repository in front of another, specified repository or to the end of the list.
This API does not move artifact version data to another repository server machine. Only the repository order in which Commander searches to retrieve an artifact version is changed.
You must specify a repositoryName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository you need to move.</td>
</tr>
<tr>
<td>beforeRepositoryName</td>
<td>Moves this repository (repositoryName) to a place before the name specified by this option. If omitted repositoryName is moved to the end.</td>
</tr>
</tbody>
</table>

**Positional arguments**
repositoryName

**Response**
Returns a modified repository element or an error if the repository does not exist.

dc-perl

**syntax:** $cmdr->moveRepository(<repositoryName>, [<optionals>]);

**Example**
$cmdr->moveRepository("reposThree", {beforeRepositoryName => "reposOne"]);

**ectool**

**syntax:** ectool moveRepository <repositoryName> ...

**Response**
Returns a modified repository element or an error if the repository does not exist.
Example

ectool moveRepository reposThree --beforeRepositoryName reposOne

publishArtifactVersion

Publishes an artifact version to an artifact repository.

Note: This API wraps the "publish" function in the ElectricCommander::ArtifactManagement Perl module and hides some additional functionality implemented in that module.

You must specify an artifactName or a groupId with an artifactKey.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactKey</td>
<td>User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of an artifact.</td>
</tr>
<tr>
<td>compress</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>dependentArtifactVersions</td>
<td>One or more artifact version queries. The most current match of each query is retrieved when the primary artifact is retrieved. Dependent artifact version query strings are in this form: &lt;groupId&gt;:&lt;artifactKey&gt;:&lt;versionRange&gt;(versionRange is optional). The version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>excludePatterns</td>
<td>Semi-colon delimited list of file-path patterns indicating which files/directories under &quot;fromDirectory&quot; to exclude when publishing an artifact version. Defaults to &quot;empty,&quot; which means no files are excluded. See more information on &quot;pattern syntax&quot; below.</td>
</tr>
</tbody>
</table>
## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>followSymlinks</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>fromDirectory</td>
<td>The directory containing files to publish as the artifact version. A subset of files can be published based on includePatterns and excludePatterns.</td>
</tr>
<tr>
<td>groupId</td>
<td>A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td>includePatterns</td>
<td>Semi-colon delimited list of file-path patterns indicating which files/directories under &quot;fromDirectory&quot; to publish in the artifact version. Defaults to &quot;empty,&quot; which means all files will be included. Conversely, if only two files are &quot;included,&quot; no other files except those two will be included. See more information on &quot;pattern syntax&quot; below.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository where you want to publish.</td>
</tr>
<tr>
<td>version</td>
<td>Unique identifier for the artifact version in the form: major.minor.patch-qualifier-buildNumber. major, minor, patch, and buildNumber are integers and qualifier can contain any character except the following: <code>\:&lt;&gt;!/?*</code> If a version argument is provided, but does not follow the above format, the version will be considered <code>0.0.0-&lt;user-specified-version-arg&gt;-0</code> implicitly. See examples below.</td>
</tr>
</tbody>
</table>

### Version number examples

<table>
<thead>
<tr>
<th>User Input</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Major.Minor.Patch: 1.0.0 Qualifier: None Build Number: 0</td>
</tr>
<tr>
<td>1.0</td>
<td>Major.Minor.Patch: 1.0.0 Qualifier: None Build Number: 0</td>
</tr>
<tr>
<td>1.0-frank</td>
<td>Major.Minor.Patch: 1.0.0 Qualifier: frank Build Number: 0</td>
</tr>
<tr>
<td>1.0-36</td>
<td>Major.Minor.Patch: 1.0.0 Qualifier: None Build Number: 36</td>
</tr>
<tr>
<td>1.0-frank-36</td>
<td>Major.Minor.Patch: 1.0.0 Qualifier: frank Build Number: 36</td>
</tr>
</tbody>
</table>
Pattern syntax
Include / exclude patterns are expressed as relative paths under the fromDirectory.

Pattern syntax and behavior is the same as Ant and uses the following wildcard specifiers:

? - matches a single character
* - matches any number of characters, but only at a single directory level
** - matches any number of directory levels

Examples:
Use *.txt to match any .txt file in the top-level directory.
Use */*.txt to match any .txt file in any child directory.
Use **/*.txt to match any .txt file at any level.

Positional arguments
None

Response
One artifactVersion element.

ec-perl
syntax: $cmdr->publishArtifactVersion({<optionals>});

Example
# Add version 1.0.0-55 for artifact myGroup:myKey with a dependency on cmdr:SDK:1.2 .0,
# and the most current version of core:infra that is greater than or equal to 2.1. 0.
# Note: In the Perl API, the argument must be specified as singular even though it
# can take multiple values.

$cmdr->publishArtifactVersion({artifactName => "myGroup:myKey",
    version => "1.0.0-55",
    dependentArtifactVersion => ["cmdr:SDK:1.2.0", "core:infra:{2.1,}"]});

ectool
syntax: ectool publishArtifactVersion ...

Example
ectool publishArtifactVersion --artifactName "myGroup:myKey" --version "1.0.0-55"

Back to Top

removeDependentsFromArtifactVersion
Removes a list of dependent artifact versions from an existing artifact version.

You must specify the artifactVersionName.
**API commands - Artifact Management**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version from which you want to remove dependents. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>dependentArtifactVersions</td>
<td>One or more artifact version queries. The most current match of each query is retrieved when the primary artifact is retrieved. Dependent artifact version query strings are in this form: &lt;groupId&gt;:&lt;artifactKey&gt;:&lt;versionRange&gt; (versionRange is optional). The version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.</td>
</tr>
</tbody>
</table>

**Positional arguments**

artifactVersionName

**Response**

None or status OK message.

**ec-perl**

*syntax:* $cmdr->removeDependentsFromArtifactVersion (<artifactVersionName>, <optionals>);

*Example*

# Note: In the Perl API, the argument must be specified as singular
# even though it can take multiple values.

$cmdr->removeDependentsFromArtifactVersion (myGroup:myKey:1.0.0-55, {dependentArtifactVersion => ["cmdr:onlineHelp:1.0.0"]});

**ectool**

*syntax:* ectool removeDependentsFromArtifactVersion <artifactVersionName> ...

*Example*

ectool removeDependentsFromArtifactVersion myGroup:myKey:1.0.0-55 --dependentArtifactVersions "cmdr:onlineHelp:1.0.0"

**retrieveArtifactVersions**

Retrieves the most recent artifact version (including its dependents) from an artifact repository.

**Note:** This API wraps the "retrieve" function in the ElectricCommander::ArtifactManagement Perl module and hides some additional functionality implemented in that module.
You must specify search criteria options to locate the artifact versions you want to retrieve.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactKey</td>
<td>User-specified identifier for this artifact. This field is limited to</td>
</tr>
<tr>
<td></td>
<td>alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version.</td>
</tr>
<tr>
<td>cacheDirectory</td>
<td>The directory where the artifact version is stored.</td>
</tr>
<tr>
<td></td>
<td>Note: The artifact version files are stored in a subdirectory under this</td>
</tr>
<tr>
<td></td>
<td>cache directory.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>filters</td>
<td>A list of zero or more filter criteria definitions used to define objects to find. Each element of the filter list is a hash reference containing one filter criterion. You may specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</td>
</tr>
</tbody>
</table>

**Two types of filters:**
- "property filters" are used to select objects based on the value of the object's intrinsic or custom property.
- "boolean filters" ("and", "or", "not") are used to combine one or more filters using boolean logic.

Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property.

**Property filter operators are:**
- between (2 operands)
- contains (1)
- equals (1)
- greaterOrEqual (1)
- greaterThan (1)
- in (1)
- lessOrEqual (1)
- lessThan (1)
- like (1)
- notEqual (1)
- notLike (1)
- isNotNull (0)
- isNull (0)

A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter.

**Boolean operators are:**
- not (1 operand)
- and (2 or more operands)
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>or (2 or more operands)</td>
</tr>
<tr>
<td>groupId</td>
<td>A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td>includeDependents</td>
<td>Options are:</td>
</tr>
<tr>
<td></td>
<td>• 0/false – dependent artifacts are not retrieved.</td>
</tr>
<tr>
<td></td>
<td>• 1/true – dependent artifacts are retrieved.</td>
</tr>
<tr>
<td>overwrite</td>
<td>Options are:</td>
</tr>
<tr>
<td></td>
<td>• true – deletes previous content in the directory and replaces the content with your new version.</td>
</tr>
<tr>
<td></td>
<td>• false – (existing behavior) if the directory does not exist, one will be created and filled with the artifact’s content. If the directory exists, a new directory is created with a unique name and the artifact contents is supplied there.</td>
</tr>
<tr>
<td></td>
<td>• update - this is similar to a merge operation—two artifact versions can be moved into the same directory, but individual files with the same name will be overwritten.</td>
</tr>
<tr>
<td>repositoryNames</td>
<td>A space-separated list of artifact repository names. Retrieval is attempted from each specified repository in a specified order until it succeeds or all specified repositories have rejected the retrieval. If not specified, and if this request is made in a job step context, a preferred list of repository names is obtained from the Resource definition in the server. If that list is empty, the global repository list is used.</td>
</tr>
<tr>
<td>toDirectory</td>
<td>Used to retrieve an artifact version to a specific directory without imposing the structure of a cache directory. Specify the full path to the new directory.</td>
</tr>
<tr>
<td></td>
<td>• If the artifact version is in a local cache directory, it will be copied out of the cache.</td>
</tr>
<tr>
<td></td>
<td>• If the artifact version is not in a cache directory, it will be downloaded directly to the specified directory, without putting it into a cache. toDirectory overrides cacheDirectory for downloads.</td>
</tr>
<tr>
<td>versionRange</td>
<td>The range of versions to search. Version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.</td>
</tr>
</tbody>
</table>

### Positional arguments

None
Response
Returns one or more artifactVersion elements.

ec-perl

syntax: $cmdr->retrieveArtifactVersions {<optionals>});

Examples
# Retrieve the most current core:infra artifact version whose version is 1.x.x.
$cmdr->retrieveArtifactVersions({groupId => "core",
    artifactKey => "infra",
    versionRange => "[1.0,2.0)"});

# Or alternatively...
$cmdr->retrieveArtifactVersions({artifactName => "core:infra",
    versionRange => "[1.0,2.0)"});

ectool

syntax: ectool retrieveArtifactVersions ...

Example
ectool retrieveArtifactVersions --artifactName "core:infra" --versionRange "[1.0,2.0)"

Note: The filter option does not perform as expected if using ectool. If you need the filter option, write your retrieveArtifactVersions API call in ec-perl.

Back to Top
API commands - Component

**addComponentToApplicationTier**

Adds the given component to the given application tier.

You must specify the `projectName`, `applicationName`, and `applicationTierName` arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>applicationTierName</code></td>
<td>Name of the tier; must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>componentName</code></td>
<td>Name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>componentProjectName</code></td>
<td>(Optional) Project name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

Returns the component and specified application tier elements.

**ec-perl**

Syntax:

```
$<object>->addComponentToApplicationTier(<projectName>, <applicationName>,
<applicationTierName>, <componentProjectName>, <componentName>,
{<optionals>});
```

Example:

```
$ec->addComponentToApplicationTier("default", "newApp", "appTier1",
"component1");
```
 ectool
Syntax:
    ectool addComponentToApplicationTier <projectName> <applicationName>
        <applicationTierName> <componentName> [optionals...]
Example:
    ectool addComponentToApplicationTier default newApp appTier1 VCScomponent

createComponent

Creates a new component for a project.
You must specify the projectName, applicationName, and applicationTierName arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>componentName</td>
<td>Name of the component.</td>
</tr>
<tr>
<td>pluginName</td>
<td>Name of the plugin.</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) Name of an application to scope this component to.</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) Name of a credential to attach to this component.</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricCommander.</td>
</tr>
</tbody>
</table>

Response

Returns a version control component element.

dc-perl
Syntax:
    $<object>-createComponent(<projectName>, <componentName>, <pluginName>,
        {<optionals>});
Example:
```perl
$ec->createComponent("default", "component1", "Publish Artifact Version",
    {description => "New agent"});
```

**ectool**

Syntax:

```bash
ectool createComponent <projectName> <componentName> <pluginName>
    [optionals...]
```

Example:

```bash
ectool createComponent default component1 "Publish Artifact Version"
    --description "New agent"
```

**deleteComponent**

Deletes a component.

You must specify the **projectName** and **componentName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>Name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of an application to which this component is scoped.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

None or a status OK message.

**ec-perl**

Syntax:

```perl
$obj->deleteComponent(<projectName>, <componentName>),
    (<optionals>);
```

Example:

```perl
$obj->deleteComponent("default", "VCScomponent");
```

**ectool**

Syntax:
ectool deleteComponent <projectName> <componentName>  
[optionals...]

Example:
ectool deleteComponent default VCScomponent

Back to Top

**getComponent**

Finds a component by name.

You must specify the **projectName** and **componentName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| projectName      | Name for the project; must be unique among all projects.  
Argument Type: String |
| componentName    | Name of the component.  
Argument Type: String |
| applicationName  | (Optional) Name of an application to which this component is scoped.  
Argument Type: String |

**Response**

Retrieves the specified component element.

**ec-perl**

Syntax:

```
$<object>->getComponent(<projectName>, <componentName>, {<optionals>});
```

Example:

```
$ec->getComponent("default", "component1");
```

**ectool**

Syntax:

```
ectool getComponent <projectName> <componentName>  
[optionals...]  
```

Example:

```
ectool getComponent default VCScomponent  
```
**getComponents**

Retrieves all components in a project.

You must specify the **projectName** argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projectName</strong></td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><strong>applicationName</strong></td>
<td>(Optional) Name of the application. Specify to search for components scoped to an application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

Retrieves zero or more component elements.

**ec-perl**

Syntax:

```
$<object>-&gt;getComponents( &lt;projectName&gt;, &lt;optionals&gt; );
```

Example:

```
$ec-&gt;getComponents("default");
```

**ectool**

Syntax:

```
ectool getComponents &lt;projectName&gt; [optionals...]
```

Example:

```
ectool getComponents default
```

[Back to Top](#)

**getComponentsInApplicationTier**

Returns the list of components in an application tier.

You must specify the **projectName**, **applicationName**, and **applicationTierName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projectName</strong></td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
applicationName | Name of the application; must be unique among all projects. Argument Type: String
applicationTierName | Name of the tier; must be unique within the application. Argument Type: String

Response
Retrieves zero or more component elements in the specified application tier.

**ec-perl**
Syntax:
```perl
$<object>->getComponentsInApplicationTier(<projectName>, <applicationName>, <applicationTierName>);
```
Example:
```
$ec->getComponentsInApplicationTier("default", "newApp", "appTier1");
```

**ectool**
Syntax:
```plaintext
ectool getComponentsInApplicationTier <projectName> <applicationName> <applicationTierName>
```
Example:
```
ectool getComponentsInApplicationTier default newApp appTier1
```

**modifyComponent**
Modifies an existing component.
**You must specify the projectName and componentName arguments.**

Arguments | Descriptions
--- | ---
projectName | Name for the project; must be unique among all projects. Argument Type: String
componentName | Name of the component. Argument Type: String
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credential Name</td>
<td>(Optional) Name of the credential. Argument Type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this component; not interpreted at all by ElectricFlow. Argument Type: String</td>
</tr>
<tr>
<td>newName</td>
<td>(Optional) New name of the component. Argument Type: String</td>
</tr>
</tbody>
</table>

### Response

Retrieves an updated component element.

**ec-perl**

Syntax:

```
$<object>-modifyComponent(<projectName>, <componentName>, {<optionals>});
```

Example:

```
$ec->modifyComponent("default", "component1", {credentialName => "cred1",
newName => "NewName"});
```

**ectool**

Syntax:

```
ectool modifyComponent <projectName> <componentName> [optionals...]
```

Example:

```
ectool modifyComponent default component1 --credentialName cred1 --newName New Name
```

### removeComponentFromApplicationTier

Removes the given component from the given application tier.

You must specify the **projectName, applicationName, applicationTierName, and componentName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects. Argument Type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>Name of the application; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>Name of the tier; must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>Name of component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

### Response

None or a status OK message.

### ec-perl

**Syntax:**

```
$<object>-removeComponentFromApplicationTierOperation(<projectName>,<applicationName>,<applicationTierName>,<componentName>);
```

**Example:**

```
$ec->removeComponentFromApplicationTierOperation("default", "newApp", "appTier1", "component1");
```

### ectool

**Syntax:**

```
ectool removeComponentFromApplicationTierOperation <projectName> <applicationName> <applicationTierName> <componentName>
```

**Example:**

```
ectool removeComponentFromApplicationTierOperation default newApp appTier1 VCScomponent
```

[Back to Top]
API Commands - Credential Management

attachCredential
createCredential
deleteCredential
detachCredential
getCredential
getCredentials
getFullCredential
modifyCredential

attachCredential

Attaches a credential to a step or a schedule.

Attaching a credential allows the credential to be passed as an actual argument by a schedule or subprocedure step, or to be used in a getFullCredential call by a command step.

You must specify projectName, credentialName, and locator arguments to identify a step or a schedule.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms:</td>
</tr>
<tr>
<td></td>
<td>relative</td>
</tr>
<tr>
<td></td>
<td>(for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>absolute</td>
</tr>
<tr>
<td></td>
<td>(for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of a procedure within the &quot;named&quot; project where this credential will be attached.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains the object where this credential will be attached.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The schedule name for running one of the procedures within the &quot;named&quot; project.</td>
</tr>
<tr>
<td>stepName</td>
<td>A step name within one of the procedures contained in the &quot;named&quot; project.</td>
</tr>
</tbody>
</table>

Positional arguments

projectName, credentialName

Response

None or status OK message.
**ec-perl**

**syntax:** `$cmdr->attachCredential(<projectName>, <credentialName>, {...});`

**Example**

```
$cmdr->attachCredential("Test Proj", "Preflight User", {procedureName => "Run Build", stepName=>"Get Sources")});
```

**ectool**

**syntax:** `ectool attachCredential <projectName> <credentialName> ...

**Example**

```
ectool attachCredential "Test Proj" "Preflight User"
   --procedureName "Run Build" --stepName "Get Sources"
```

**createCredential**

Creates a new credential for a project.

You must specify a `projectName` and `credentialName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential to create (any name you choose).</td>
</tr>
<tr>
<td>password</td>
<td>The password matching the specified user name.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project where the credential will be stored.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `credentialName`

**Response**

None or status OK message.

**ec-perl**

**syntax:** `$cmdr->createCredential(<projectName>, <credentialName>, [{optionals}]);`
**Example**

```php
$cmdr->createCredential("Sample Project", "Build User", 
    [userName => "build", 
     password => "abc123"]);
```

**ectool**

**syntax:**

```bash
ectool createCredential <projectName> <credentialName> --userName <userName> 
    --password <password> ...
```

**Example**

```bash
ectool createCredential "Sample Project" "Build User" 
    --userName build --password a
    bc123
```

**deleteCredential**

Deletes a credential.

You must specify a `projectName` and a `credentialName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that contains this credential.</td>
</tr>
<tr>
<td>credentialName</td>
<td><code>credentialName</code> can be one of two forms: relative (for example, &quot;cred1&quot;) -</td>
</tr>
<tr>
<td></td>
<td>the credential is assumed to be in the project that contains the request</td>
</tr>
<tr>
<td></td>
<td>target object.</td>
</tr>
<tr>
<td></td>
<td>absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the</td>
</tr>
<tr>
<td></td>
<td>credential can be from any specified project, regardless of the target</td>
</tr>
<tr>
<td></td>
<td>object's project.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `credentialName`

**Response**

None or a status OK message.

**ec-perl**

**syntax:**

```perl
$cmdr->deleteCredential(<projectName>, <credentialName>);
```

**Example**

```perl
$cmdr->deleteCredential('Sample Project', 'Build User');
```

**ectool**

**syntax:**

```bash
ectool deleteCredential <projectName> <credentialName>
```
detachCredential

Detaches a credential from an object.

You must specify projectName and credentialName. Also, depending on where the credential is attached, you must specify a step (using procedureName and stepName), or define a schedule (using scheduleName).

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms:</td>
</tr>
<tr>
<td></td>
<td>relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the</td>
</tr>
<tr>
<td></td>
<td>project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the</td>
</tr>
<tr>
<td></td>
<td>credential can be from any specified project, regardless of the target</td>
</tr>
<tr>
<td></td>
<td>object's project.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that contains the step with the credential to</td>
</tr>
<tr>
<td></td>
<td>detach.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains this credential, schedule, or procedure</td>
</tr>
<tr>
<td></td>
<td>step.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule where this credential may be attached.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step where this credential may be attached.</td>
</tr>
<tr>
<td></td>
<td><strong>Also requires</strong> procedureName.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, credentialName

**Response**

None, or a status OK message on success, or:

NoSuchCredential if the specified credential does not exist.

NoSuchSchedule if the specified schedule does not exist.

**ec-perl**

*syntax*: $cmdr->detachCredential(<projectName>, <credentialName>, [<optionals>]);

**Examples**

$cmdr->detachCredential("Test Proj", "Preflight User",
    {procedureName => "Run Build",}
stepName => "Get Sources"});

$cmdr->detachCredential("Test Proj", "Preflight User",
   {scheduleName => "Build Schedule"});

**ectool**

**syntax:** ectool detachCredential <projectName> <credentialName> ...

**Examples**

```bash
ectool detachCredential "Test Proj" "Preflight User"
   --procedureName "Run Build" --stepName "Get Sources"

ectool detachCredential "Test Proj" "Preflight User"
   --scheduleName "Build Schedule"
```

**getCredential**

Finds a credential by name.

You must specify **projectName** and **credentialName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing this credential.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**projectName**, **credentialName**

**Response**

One **credential** element.

**ec-perl**

**syntax:** $cmdr->getCredential(<projectName>, <credentialName> );

**Example**

```perl
$cmdr->getCredential("SampleProject", "Build User");
```

**ectool**

**syntax:** ectool getCredential <projectName> <credentialName>
**Example**

ectool getCredential "Sample Project" "Build User"

---

**getCredentials**

Retrieves all credentials in a project.

You must specify a `projectName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project containing these credentials.</td>
</tr>
<tr>
<td><code>usableOnly</code></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `projectName`

**Response**

Zero or more `credential` elements.

**ec-perl**

`syntax:` `$cmdr->getCredentials(<projectName>, {...});`

*Example*

```
$cmdr->getCredentials("Sample Project", {"usableOnly" => 1});
```

**ectool**

`syntax:` `ectool getCredentials <projectName> ...`

*Example*

```
ectool getCredentials "Sample Project" --usableOnly 1
```

---

**getFullCredential**

Finds a credential by name, including password, from within a running step.

You must specify the `credentialName`. 
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>value</td>
<td>&lt;password</td>
</tr>
</tbody>
</table>

### Positional arguments

`credentialName`

### Response

If value is supplied, only the name is returned when called by ectool. If no value is supplied, an xPath object is returned.

### ec-perl

**Syntax:**

```perl
$cmdr->getFullCredential(<credentialName>, {<optionals>});
```

**Example**

```perl
my $xpath = $cmdr->getFullCredential("myCred", {value => "password"});
```

```perl
my $password = $xpath->find("//password");
```

### ectool

**Syntax:**

```ectool
getcherFullCredential <credentialName> ...
```

**Example**

```ectool
getcherFullCredential myCred --value password
```

### Back to Top

---

### modifyCredential

Modifies an existing credential.

You must specify `projectName` and `credentialName`.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, /projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the credential.</td>
</tr>
<tr>
<td>password</td>
<td>The password for the specified user name.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing this credential.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user containing this credential.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, credentialName

**Response**

None or a status OK message.

**ec-perl**

* syntax:*

```perl
$cmdr->modifyCredential(<projectName>, <credentialName>, {[<optionals>]})
```

* Example*

```perl
$cmdr->modifyCredential("Sample Project", "Build User", {userName => "build"});
```

**ectool**

* syntax:*

```bash
ectool modifyCredential <projectName> <credentialName> ...
```

* Example*

```bash
ectool modifyCredential "Sample Project" "Build User" --userName build
```

Back to Top
API Commands - Database Configuration

getDatabaseConfiguration

Retrieves the current database configuration.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Returns a `databaseConfiguration` element, which includes the database name, user name, database dialect, driver, URL, along with the host name and port number.

**ec-perl**

`syntax:` 

```perl
$cmdr->getDatabaseConfiguration();
```

**Example**

```perl
$cmdr->getDatabaseConfiguration();
```

**ectool**

`syntax:` 

`ectool getDatabaseConfiguration`

**Example**

`ectool getDatabaseConfiguration`

**setDatabaseConfiguration**

Sets the database configuration on the server. If the server is in bootstrap mode, these changes take effect immediately and the server attempts to start. If the server is already running, these changes have no effect until the server is restarted.

**Note:** If you are replacing the database you are currently using, you must restart the Commander server after configuring the new database you want to use.

ElectricCommander assigns default values to the following three arguments—these values are derived from information you supply for the arguments below. The values for these arguments can be viewed in the XML Response for `getDatabaseConfiguration`. You should not need to change these values, but "customDatabase" arguments may be used to over-ride Commander default values. Contact Electric Cloud Customer Support for assistance with using these arguments:
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>customDatabaseDialect</td>
<td>Class name of the Hibernate dialect (<em>advanced use only</em>—the server will choose an appropriate dialect based on the databaseType).</td>
</tr>
<tr>
<td>customDatabaseDriver</td>
<td>Class name of the JDBC driver (<em>advanced use only</em>—the server will choose an appropriate driver based on the databaseType).</td>
</tr>
<tr>
<td>customDatabaseUrl</td>
<td>The JDBC to use (<em>advanced use only</em>—the server will compose an appropriate URL).</td>
</tr>
<tr>
<td>databaseName</td>
<td>The name of the database you want the Commander server to use.</td>
</tr>
<tr>
<td>databaseType</td>
<td>The type of database you want the Commander server to use. Supported database types are: &lt;builtin</td>
</tr>
<tr>
<td>hostName</td>
<td>The name of the host machine where the database is running.</td>
</tr>
<tr>
<td>ignorePasskeyMismatch</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>ignoreServerMismatch</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>password</td>
<td>The password required to access the database. setDatabaseConfiguration does not allow a passwordless database user. Make sure the database user has a password.</td>
</tr>
<tr>
<td>port</td>
<td>The port number used to access the database.</td>
</tr>
<tr>
<td>preserveSessions</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user required to access the database.</td>
</tr>
</tbody>
</table>

**Positional arguments**

None
Response
None or a status OK message.

cp-perl

`syntax:` $cmdr->setDatabaseConfiguration({<optionals>});

`Example`

$cmdr->setDatabaseConfiguration({hostName => "localhost", port => 3306});
# If the database type is set to the mysql, sqlserver, or oracle and
# you want to use the builtin database

cp-perl

$cmdr->setDatabaseConfiguration({databaseType => "builtin", databaseName => "builtin"});

ectool

`syntax:` ectool setDatabaseConfiguration <specify configuration values> ...>

`Example`

ectool setDatabaseConfiguration --hostName localhost --port 3306
# If the database type is set to the mysql, sqlserver, or oracle and
# you want to use the builtin database

ectool setDatabaseConfiguration --databaseType builtin --databaseName builtin

Back to Top
API Commands - Directory Provider Management

createDirectoryProvider
deleteDirectoryProvider
getDirectoryProvider
getDirectoryProviders
modifyDirectoryProvider
moveDirectoryProvider
testDirectoryProvider

createDirectoryProvider

Creates a new Active Directory or LDAP directory provider.

You must specify a **providerName**, **providerType**, and **url**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>commonGroupNameAttribute</td>
<td>The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider. Use this argument if the <strong>groupNameAttribute</strong> or the <strong>uniqueGroupNameAttribute</strong> is set to <strong>distinguishedName</strong>, which is not searchable.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code>.</td>
</tr>
<tr>
<td>domainName</td>
<td>The domain name from which Active Directory server(s) are automatically discovered.</td>
</tr>
<tr>
<td>emailAttribute</td>
<td>The attribute in a user record that contains the user's email address. If the attribute is not specified, the account name and domain name are concatenated to form an email address.</td>
</tr>
<tr>
<td>enableGroups</td>
<td>`&lt;Boolean flag -0</td>
</tr>
<tr>
<td>fullUserNameAttribute</td>
<td>The attribute in a user record that contains the user’s full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.</td>
</tr>
<tr>
<td>groupBase</td>
<td>This string is prepended to the <strong>basedn</strong> to construct the directory DN that contains group records.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>groupMemberAttributes</td>
<td>A comma-separated attribute name list that identifies a group member. Most LDAP configurations only specify a single value, but if there is a mixture of POSIX and LDAP style groups in the directory, multiple attributes might be required.</td>
</tr>
<tr>
<td>groupMemberFilter</td>
<td>This LDAP query is performed in the groups directory context to identify groups containing a specific user as a member. Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and groupOfNames or uniqueGroupOfNames records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: &quot;0&quot; is replaced with the full user record DN, and &quot;1&quot; is replaced with the user's account name.</td>
</tr>
<tr>
<td>groupNameAttribute</td>
<td>The group record attribute that contains the name of the group.</td>
</tr>
<tr>
<td>groupSearchFilter</td>
<td>This LDAP query is performed in the context of the groups directory to enumerate group records.</td>
</tr>
<tr>
<td>managerDn</td>
<td>The DN of a user who has read-only access to LDAP user and group directories. If this property is not specified, the server attempts to connect as an unauthenticated user. Not all servers allow anonymous read-only access. <strong>Note:</strong> This user does not need to be an admin user with modify privileges.</td>
</tr>
<tr>
<td>managerPassword</td>
<td>If the <em>managerDn</em> property is set, this password is used to authenticate the manager user.</td>
</tr>
<tr>
<td>providerName</td>
<td>This human-readable name will be displayed in the user interface to identify users and groups that come from this provider.</td>
</tr>
<tr>
<td>providerType</td>
<td>`&lt;ldap</td>
</tr>
<tr>
<td>realm</td>
<td>This is an identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in &quot;same name&quot; collisions across multiple directory providers. The realm is appended to the user or group name when stored in the Commander server. For example, <code>&lt;user&gt;@dir</code> (where the realm is set to &quot;dir&quot;).</td>
</tr>
<tr>
<td>url</td>
<td>The server URL is in the form <code>protocol://host:port/basedn</code>. Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for <code>ldap</code>, 636 for <code>ldaps</code>). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a <code>dc=</code> and separated by commas. <strong>Note:</strong> Spaces in the <code>basedn</code> must be URL encoded (<code>%20</code>).</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>userBase</td>
<td>This string is prepended to the basedn to construct the directory DN that contains user records.</td>
</tr>
<tr>
<td>userNameAttribute</td>
<td>The attribute in a user record that contains the user's account name.</td>
</tr>
<tr>
<td>userSearchFilter</td>
<td>This LDAP query is performed in the context of the user directory to search for a user by account name. The string &quot;{0}&quot; is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.</td>
</tr>
<tr>
<td>userSearchSubtree</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>useSSL</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

providerName, providerType, url

**Response**

None or status OK message.

**ec-perl**

```perl
syntax: $cmdr->createDirectoryProvider(<providerName>, {<optionals>});
```

**Example**

```perl
$cmdr->createDirectoryProvider("AD3", {url => "ldaps://pdc/dc=coname3.dc=com", providerType => "activedirectory");
```

**ectool**

```bash
syntax: ectool createDirectoryProvider <providerName> ...
```

**Example**

```bash
ectool createDirectoryProvider AD3 --url "ldaps://pdc/dc=coname3.dc=com" --providerType activedirectory
```

---

**deleteDirectoryProvider**

Deletes an Active Directory or LDAP directory provider.

You must specify a providerName.
### deleteDirectoryProvider

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerName</td>
<td>The name of the directory provider you want to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- **providerName**

**Response**

None or a status OK message.

**ec-perl**

**syntax:** `$cmdr->deleteDirectoryProvider(<providerName>);`

**Example**

```perl
$cmdr->deleteDirectoryProvider('AD3');
```

**ectool**

**syntax:** `ectool deleteDirectoryProvider <providerName>`

**Example**

```bash
ectool deleteDirectoryProvider AD3
```

### getDirectoryProvider

**getDirectoryProvider**

Retrieves a directory provider by name.

You must specify a **providerName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerName</td>
<td>The name of the directory provider.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- **providerName**

**Response**

One **directoryProvider** element.

**Note:** For security reasons, the **managerPassword** field is never returned.

**ec-perl**

**syntax:** `$cmdr->getDirectoryProvider(<providerName>);`

**Example**

```perl
$cmdr->getDirectoryProvider("AD3");
```
**ectool**

*Syntax:* `ectool getDirectoryProvider <providerName>`

*Example*

`ectool getDirectoryProvider AD3`

**getDirectoryProviders**

Retrieves all directory providers.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Zero or more *directoryProvider* elements.

**ec-perl**

*Syntax:* `$cmdr->getDirectoryProviders();`

*Example*

```
$cmdr->getDirectoryProviders();
```

**ectool**

*Syntax:* `ectool getDirectoryProviders`

*Example*

`ectool getDirectoryProviders`

**modifyDirectoryProvider**

Modifies an existing LDAP directory provider.

You must specify the *providerName*. 
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commonGroupNameAttribute</td>
<td>The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider. Use this argument if the groupNameAttribute or the uniqueGroupNameAttribute is set to distinguishedName, which is not searchable.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tt&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td>domainName</td>
<td>The domain from which Active Directory servers are automatically discovered.</td>
</tr>
<tr>
<td>emailAttribute</td>
<td>The attribute in a user record that contains the user's email address. If the attribute is not specified, the account name and domain name are concatenated to form an email address.</td>
</tr>
<tr>
<td>enableGroups</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>fullUserNameAttribute</td>
<td>The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.</td>
</tr>
<tr>
<td>groupBase</td>
<td>This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records.</td>
</tr>
<tr>
<td>groupMemberAttributes</td>
<td>A comma-separated attribute name list that identifies a group member. Most LDAP configurations only specify a single value, but if there is a mixture of POSIX and LDAP style groups in the directory, multiple attributes might be required.</td>
</tr>
<tr>
<td>groupMemberFilter</td>
<td>This LDAP query is performed in the group directory context to identify groups containing a specific user as a member. Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and groupOfNames or uniqueGroupOfNames records where members are identified by the full user DN. Both forms are supported, so the query is passed two parameters: &quot;{0}&quot; is replaced with the full user record DN, and &quot;(1)&quot; is replaced with the user's account name.</td>
</tr>
<tr>
<td>groupNameAttribute</td>
<td>The group record attribute that contains the name of the group.</td>
</tr>
<tr>
<td>groupSearchFilter</td>
<td>A filter name: this LDAP query is performed in the context of the groups directory to enumerate group records.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>managerDn</td>
<td>The DN of a user who has read access to LDAP user and group directories. If this property is not specified, the server attempts to connect as an unauthenticated user. Not all servers allow anonymous read-only access. <strong>Note:</strong> This user does not need to be an admin user with modify privileges.</td>
</tr>
<tr>
<td>managerPassword</td>
<td>If the <code>managerDn</code> property is set, this password is used to authenticate the manager user.</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the directory provider.</td>
</tr>
<tr>
<td>providerName</td>
<td>This human readable name will be displayed in the user interface to identify users and groups that come from this provider.</td>
</tr>
<tr>
<td>providerType</td>
<td>`&lt;ldap</td>
</tr>
<tr>
<td>realm</td>
<td>This is an identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in &quot;same name&quot; collisions across multiple directory providers. The <code>realm</code> is appended to the user or group name when stored in the Commander server. For example, <code>&lt;user&gt;@dir</code> (where the <code>realm</code> is set to &quot;dir&quot;).</td>
</tr>
<tr>
<td>url</td>
<td>The LDAP server URL is in the form <code>protocol://host:port/basedn</code>. Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for <code>ldap</code>, 636 for <code>ldaps</code>). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a <code>dc=</code> and separated by commas. <strong>Note:</strong> Spaces in the <code>basedn</code> must be URL encoded (%20).</td>
</tr>
<tr>
<td>userBase</td>
<td>This string is prepended to the <code>basedn</code> to construct the directory DN that contains user records.</td>
</tr>
<tr>
<td>userNameAttribute</td>
<td>The attribute in a user record that contains the user's account name.</td>
</tr>
<tr>
<td>userSearchFilter</td>
<td>This LDAP query is performed in the context of the user directory to search for a user by account name. The string &quot;{0}&quot; is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.</td>
</tr>
<tr>
<td>userSearchSubtree</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
useSSL | <Boolean flag - 0|1|true|false> Use this flag to define whether or not SSL is used for server-agent communication, or if you need to use SSL to communicate with your Active Directory servers. Default is "true".

**Positional arguments**

**providerName**

Response

None or a status OK message.

**ec-perl**

*Syntax:* $cmdr->modifyDirectoryProvider(<providerName>, [<optionals>]);

*Example*

$cmdr->modifyDirectoryProvider("AD3", {emailAttribute => "email"});

**ectool**

*Syntax:* ectool modifyDirectoryProvider <providerName> ...

*Example*

ectool modifyDirectoryProvider AD3 --emailAttribute email

**moveDirectoryProvider**

Moves an Active Directory or LDAP directory provider in front of another specified provider or to the end of the list.

You must specify a providerName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerName</td>
<td>The name of the directory provider to move.</td>
</tr>
<tr>
<td>beforeProviderName</td>
<td>Moves this directory provider (providerName) to a place before the name specified by this option. If omitted, providerName is moved to the end.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**providerName**

Response

None or a status OK message.
API Commands - Directory Provider Management

**ec-perl**

*syntax:* $cmdr->moveDirectoryProvider(<providerName>, {<optionals>});

*Example*

$cmdr->moveDirectoryProvider("AD3", {beforeProviderName => "AD2"});

**ectool**

*syntax:* ectool moveDirectoryProvider <providerName> ...

*Example*

ectool moveDirectoryProvider AD3 --beforeProviderName AD2

Back to Top

**testDirectoryProvider**

Tests that a specific user name and password combination work with the specified directory provider settings. You must specify *userName* and *password* (the command will prompt for the password if it is omitted).

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>commonGroupNameAttribute</td>
<td>The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider. Use this argument if the groupNameAttribute or the uniqueGroupNameAttribute is set to distinguishedName, which is not searchable.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>domainName</td>
<td>The domain from which Active Directory servers are automatically discovered.</td>
</tr>
<tr>
<td>emailAttribute</td>
<td>The attribute in a user record that contains the user's email address. If the attribute is not specified, the account name and domain name are concatenated to form an email address.</td>
</tr>
<tr>
<td>enableGroups</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>fullUserNameAttribute</td>
<td>The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>groupBase</td>
<td>This string is prepended to the basedn to construct the directory DN that contains group records.</td>
</tr>
<tr>
<td>groupMemberAttributes</td>
<td>A comma separated attribute name list that identifies a group member. Most LDAP configurations only specify a single value, but if there is a mixture of POSIX and LDAP style groups in the directory, multiple attributes might be required.</td>
</tr>
<tr>
<td>groupMemberFilter</td>
<td>This LDAP query is performed in the groups directory context to identify groups containing a specific user as a member. Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and groupOfNames or uniqueGroupOfNames records where members are identified by the full user DN. Both forms are supported, so the query is passed two parameters: &quot;{0}&quot; is replaced with the full user record DN, and &quot;{1}&quot; is replaced with the user's account name.</td>
</tr>
<tr>
<td>groupNameAttribute</td>
<td>The group record attribute that contains the name of the group.</td>
</tr>
<tr>
<td>groupSearchFilter</td>
<td>This LDAP query is performed in the context of the groups directory to enumerate group records.</td>
</tr>
<tr>
<td>managerDn</td>
<td>The DN of a user who has read-only access to LDAP user and group directories. If this property is not specified, the server attempts to connect as an unauthenticated user. Not all servers allow anonymous read-only access. <strong>Note:</strong> This user does not need to be an admin user with modify privileges.</td>
</tr>
<tr>
<td>managerPassword</td>
<td>If the managerDn property is set, this password is used to authenticate the manager user.</td>
</tr>
<tr>
<td>password</td>
<td>The password for the user that you are testing for this provider. The command will prompt for the password if it is omitted.</td>
</tr>
<tr>
<td>providerType</td>
<td>`&lt;ldap</td>
</tr>
<tr>
<td>realm</td>
<td>This is an identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in &quot;same name&quot; collisions across multiple directory providers. The realm is appended to the user or group name when stored in the Commander server. For example, <code>&lt;user&gt;@dir (where the realm is set to &quot;dir&quot;).</code></td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
url | The LDAP server URL is in the form `protocol://host:port/basedn`. Protocol is either ldap or ldaps (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for ldap, 636 for ldaps). The basedn is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a `dc=` and separated by commas. **Note:** Spaces in the basedn must be URL encoded (%20).

useDefaults | `<Boolean flag - 0|1|true|false>` If “true”, defaults will be used for all fields not specified.

userBase | This string is prepended to the base DN to construct the directory DN that contains user records.

userName | The name of the user you are testing for this provider.

userNameAttribute | The attribute in a user record that contains the user's account name.

userSearchFilter | A filter name. This LDAP query is performed in the context of the user directory to search for a user by account name. The string "{0}" is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.

userSearchSubtree | `<Boolean flag - 0|1|true|false>` If "true", recursively search the subtree below the user base.

useSSL | `<Boolean flag - 0|1|true|false>` Use this flag to define whether or not SSL is used for server-agent communication, or if you need to use SSL to communicate with your Active Directory servers. Default is "true".

**Positional arguments**

*userName*, *password*

**Response**

Three queries are returned: One query authenticates the user *userAuthenticationTest*, one query retrieves information about the user *findUserTest*, and one shows the results of finding groups where the user is a member *findGroupsTest*.

**ec-perl**

**syntax:** `$cmdr->testDirectoryProvider(<userName>, <password>, {<optionals>});`

**Example**

```
$cmdr->testDirectoryProvider("testUser", "testUserPassword", 
{providerType => "activedirectory", 
  domainName => "my-company.com"},
```
useDefaults => 1,
managerDn => "testManager",
managerPassword => "testManagerPassword"});

**ectool**

*Syntax:*

`ectool testDirectoryProvider <userName> <password> ...
``

**Example**

```bash
ectool testDirectoryProvider testUser testUserPassword --providerType activeDirectory
   --domainName my-company.com
   --useDefaults 1
   --managerDn testManager
   --managerPassword testManagerPassword
```

[Back to Top](#)
**createEmailConfig**

Creates a new email configuration.

You must specify **configName, mailFrom, and mailHost**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>configName</strong></td>
<td>The name of your email configuration.</td>
</tr>
<tr>
<td><strong>description</strong></td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tt&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td><strong>mailFrom</strong></td>
<td>The email address used as the email sender address for notifications.</td>
</tr>
<tr>
<td><strong>mailHost</strong></td>
<td>The name of the email server host.</td>
</tr>
<tr>
<td><strong>mailPort</strong></td>
<td>The port number for the mail server, but may not need to be specified. The protocol software determines the default value (25 for SMTP and 465 for SSMTP). Specify a value for this argument when a non-default port is used.</td>
</tr>
<tr>
<td><strong>mailProtocol</strong></td>
<td>This is either SSMTP or SMTP (not case-sensitive). The default is SMTP.</td>
</tr>
<tr>
<td><strong>mailUser</strong></td>
<td>This can be an individual or a generic name like &quot;Commander&quot; - name of the email user on whose behalf Commander sends email notifications.</td>
</tr>
<tr>
<td><strong>mailUserPassword</strong></td>
<td>Password for the email user who is sending notifications.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`configName`

**Response**

None or status OK message.
**deleteEmailConfig**

Deletes an email configuration.

You must specify a `configName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>configName</code></td>
<td>The name of the email configuration you want to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`configName`

**Response**

None or a status OK message.

**ec-perl**

**syntax:**

```
$cmdr->createEmailConfig(<configName>, {<optionals>});
```

**Example**

```
$cmdr->createEmailConfig("testConfiguration",
  {mailHost => "ectest-sol2",
   mailFrom => 'commander@electric-cloud.com',
   mailUser => "build@electric-cloud.com",
   mailUserPassword => "mybuildmail"});
```

**ectool**

**syntax:**

```
createEmailConfig <configName> ...
```

**Example**

```
ectool createEmailConfig EmailConfig_test --mailHost ectest-sol2
   --mailFrom commander@electric-cloud.com --mailUser "build@electric-cloud.com"
   --mailUserPassword "mybuildmail" --description "This is a test for the email config object"
```

**deleteEmailConfig**

**syntax:**

```
$cmdr->deleteEmailConfig(<configName>);
```

**Example**

```
$cmdr->deleteEmailConfig("emailA");
```

**ectool**

**syntax:**

```
deleteEmailConfig <configName>
```

**Example**

```
ectool deleteEmailConfig emailA
```
getEmailConfig

Retrieves an email configuration by name.
You must specify a configName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>configName</td>
<td>The name of the email configuration.</td>
</tr>
</tbody>
</table>

**Positional arguments**
configName

**Response**
Returns one emailConfig element.

*Note: The mailUserPassword attribute value is not returned or displayed by the getEmailConfigs and getEmailConfig commands for security reasons.*

**ec-perl**

*Syntax:* $cmdr->getEmailConfig(<configName>);

*Example*

```perl
$cmdr->getEmailConfig("EmailConfig_test");
```

**ectool**

*Syntax:* ectool getEmailConfig <configName>

*Example*

```bash
ectool getEmailConfig EmailConfig_test
```

getEmailConfigs

Retrieves all email configurations.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

**Positional arguments**
None
Response

Returns one or more emailConfig elements.

Notes:
1. The mailUserPassword attribute value is not returned or displayed by the getEmailConfigs and getEmailConfig commands for security reasons.

2. The configIndex attribute is managed internally by ElectricCommander and cannot be used in any of the email configuration APIs. It is used internally to identify the order of emailConfig objects within the list.

ec-perl

syntax: $cmdr->getEmailConfigs();

Example

$cmdr->getEmailConfigs();

ectool

syntax: ectool getEmailConfigs

Example

ectool getEmailConfigs

modifyEmailConfig

Modifies an existing email configuration.

You must specify the configName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>configName</td>
<td>The name of your email configuration.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object.</td>
</tr>
<tr>
<td>mailFrom</td>
<td>The email address used as the email &quot;sender&quot; address for notifications.</td>
</tr>
<tr>
<td>mailHost</td>
<td>The name of the email server host.</td>
</tr>
</tbody>
</table>

Back to Top
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>mailPort</td>
<td>The port number for the mail server, but may not need to be specified. The protocol software determines the default value (25 for SMTP and 465 for SSMTP). Specify a value for this argument when a non-default port is used.</td>
</tr>
<tr>
<td>mailProtocol</td>
<td>This is either SSMTP or SMTP (not case-sensitive). Default is SMTP.</td>
</tr>
<tr>
<td>mailUser</td>
<td>The name of the email user, which can be an individual or a generic name like &quot;Commander&quot;.</td>
</tr>
<tr>
<td>mailUserPassword</td>
<td>The password for the email user.</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the email configuration.</td>
</tr>
</tbody>
</table>

**Positional arguments**

configName

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `$cmdr->modifyEmailConfig(<configName>, {<optionals>});`

*Example*

```perl
$cmdr->modifyEmailConfig("testConfiguration",
    {mailFrom => "test@my-company.com"});
```

**ectool**

*Syntax:* `ectool modifyEmailConfig <configName> ...

*Example*

`ectool modifyEmailConfig testconfiguration --mailFrom test@my-company.com --description "This is a Secure SMTP email config object for testing"`

Back to Top
API Commands - Email Notifier Management

createEmailNotifier
deleteEmailNotifier
getEmailNotifier
getEmailNotifiers
modifyEmailNotifier
sendEmail

createEmailNotifier

Creates an email notifier attached to the specified object.
You must specify a notifierName and object locators for either a job, job step, procedure, or procedure step.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>Only send mail if the condition evaluates to &quot;true&quot;. The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is &quot;false&quot; or &quot;0&quot;. If no condition is specified, the notification is ALWAYS sent.</td>
</tr>
<tr>
<td>configName</td>
<td>If specified, this argument must specify the name of an emailConfig object. If not specified, the default value is the name of the FIRST emailConfig object defined for the Commander server (emailConfig objects are &quot;ordered&quot; Commander entities). <strong>Note:</strong> If using this argument, you must include either the formattingTemplate or the formattingTemplateFile argument also.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>destinations</td>
<td>A mandatory argument for a create operation. A space-separated list of valid email addresses, email aliases, or Commander user names, or a string subject to property expansion that expands into such a list.</td>
</tr>
</tbody>
</table>
| eventType         | <onStart|onCompletion>
"onStart" triggers an event when the job or job step begins.
"onCompletion" triggers an event when the job finishes, no matter how it finishes. Default is "onCompletion." |
| formattingTemplate | This argument specifies a template for formatting email messages when an event notification is triggered by the emailNotifier. Make sure the content is formatted correctly, i.e., no illegal characters or spacing. |
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>formattingTemplateFile</td>
<td>This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the <em>formatting template file</em> is read and stored in the &quot;formatting template&quot; field. This is an alternative argument for --formattingTemplate and is useful if the &quot;formatting template&quot; field spans multiple lines.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project. <strong>Also requires</strong> procedureName</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure. <strong>Also requires</strong> projectName</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step. <strong>Also requires</strong> projectName and procedureName</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

### Positional arguments

- notifierName

### Response

None or status OK message.

### ec-perl

**syntax:**

```
$cmdr->createEmailNotifier(<notifierName>, {<optionals>});
```

**Example**

```perl
$cmdr->createEmailNotifier("testNotifier",
    {eventType => "onStart",
     condition => "$[/javascript if(myJobStep.outcome == 'warning') 'true'; else 'false'];",
     destinations => 'user1@abc.com user2@abc.com emailAlias1@abc.com',
```
configName => "testConfiguration",
projectName => "Project_test",
procedureName => "Procedure_test",
formattingTemplate => "Subject: Job started Notification: Job: ${/myJob/jobName} ${/myEvent/type}
Job: ${/myJob/jobName} ${/myEvent/type} at ${/myEvent/time}";

ectool

syntax: ectool createEmailNotifier <notifierName> ...

Example

ectool createEmailNotifier testNotifier --condition "${/javascript if(myJobStep.outcome == 'warning') 'true'; else 'false';}"
--destinations "user1@abc.com user2@abc.com emailAlias1@abc.com"
--configName EmailConfig_test --formattingTemplate "Notification: Job: ${/myJob/jobName} ${/myEvent/type} at ${/myEvent/time}"
--procedureName Procedure_test
--description "This is a test email notifier for Job completion"

deleteEmailNotifier

Deletes an email notifier from a procedure, procedure step, job, or job step.

You must specify a notifierName, and you must specify locator arguments to find the email notifier you want to delete.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier you want to delete.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that contains this email notifier. Also requires projectName</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains this email notifier.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
</tbody>
</table>
### deleteEmailNotifier

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepName</td>
<td>The name of the step that contains this email notifier.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- notifierName

**Response**

None or a status OK message.

**ec-perl**

**syntax:** $cmdr->deleteEmailNotifier('<notifierName>', { ...});

**Example**

```perl
$cmdr->deleteEmailNotifier(emailNotifier_stepTest, {projectName => "Project_test", procedureName => "Procedure_test", stepName => "Step_test2"});
```

**ecTool**

**syntax:** ecTool deleteEmailNotifier <notifierName> ...

**Example**

```bash
ectool deleteEmailNotifier emailNotifier_stepTest --projectName Project_test
   --procedureName Procedure_test --stepName Step_test2
```

### getEmailNotifier

Retrieves an email notifier from a property sheet container.

You must specify a notifierName and object locators to identify the object where the notifier is attached.

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of your email notifier.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| procedureName           | The name of the procedure.  
  Also requires the projectName |
| projectName             | The name of the project that contains this email notifier.  
  Also requires the procedureName |
| stateDefinitionName     | The name of the state definition.                                           |
| stateName               | The name of the state.                                                      |
| stepName                | The name of the step.  
  Also requires the procedureName and the projectName |
| workflowDefinitionName  | The name of the workflow definition.                                        |
| workflowName            | The name of the workflow.                                                   |

### Positional arguments

- **notifierName**

### Response

- **Returns** one emailNotifier element.

### ec-perl

- **syntax:** `$cmdr->getEmailNotifier(<notifierName>, {<optionals>});`

- **Example**

  ```perl
  $cmdr->getEmailNotifier("Error", {projectName => "Test",  
  procedureName => "Build");
  ```

### ectool

- **syntax:** `ectool getEmailNotifier <notifierName> ...`

- **Example**

  ```bash
  ectool getEmailNotifier Error --projectName Test --procedureName Build  
  --procedureName Procedure_test
  ```

### getgetEmailNotifiers

Retrieves all email notifiers defined for the specified property sheet container.

- **You must specify one or more object locators.**
### API Commands - Email Notifier Management

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the email notifier. <strong>Also requires the</strong> projectName <strong>parameter</strong>.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the email notifier. <strong>Also requires the</strong> procedureName <strong>parameter</strong>.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step. <strong>Also requires the</strong> procedureName <strong>and the</strong> projectName <strong>parameters</strong>.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**
- Arguments to locate the notifier, beginning with the top-level object locator.

**Response**
- Returns one or more `emailNotifier` elements.

**ec-perl**
- **syntax:** `$cmdr->getEmailNotifiers({<optionals>});`

**Example**
- `$cmdr->getEmailNotifiers({projectName => "Test", procedureName => "Build"});`

**ectool**
- **syntax:** `ectool getEmailNotifiers ...

**Example**
- `ectool getEmailNotifiers --projectName Project_test --procedureName Procedure_test`
modifyEmailNotifier

Modifies an email notifier in a property sheet container specified by an `emailNotifierSelector`.

**Note:** Email notifiers are evaluated and sent based on the privileges of the notifier's owner. "Owner" can be changed to the current user if that user has sufficient privileges to have deleted the notifier object and recreated it.

Modify privilege on the "admin" system ACL is required.

You must specify a `notifierName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>condition</td>
<td>Only send mail if the condition evaluates to &quot;true&quot;. The condition is a string subject to property expansion. Notification will NOT be sent if the expanded string is &quot;false&quot; or &quot;0&quot;. If no condition is specified, the notification is always sent.</td>
</tr>
<tr>
<td>configName</td>
<td>If specified, this argument must specify the name of an <code>emailConfig</code> object. If not specified, the default value is the name of the FIRST <code>emailConfig</code> object defined for the Commander server (emailConfig objects are &quot;ordered&quot; ElectricCommander entities). <strong>Note:</strong> If using this argument, you must include either <code>formattingTemplate</code> or <code>formattingTemplateFile</code> also (not both arguments).</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tr&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td>destinations</td>
<td>A space-separated list of valid email addresses, email aliases, or ElectricCommander user names, or a string subject to property expansion that expands into such a list. <strong>Note:</strong> This argument is mandatory for the &quot;create&quot; operation.</td>
</tr>
<tr>
<td>eventType</td>
<td>`&lt;onStart</td>
</tr>
<tr>
<td>formattingTemplate</td>
<td>This argument specifies a template for formatting email messages when an event [notification] is triggered by the <code>emailNotifier</code>. Make sure the content is formatted correctly, i.e., no illegal characters or spacing.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formattingTemplateFile</td>
<td>This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the formatting template file is read and stored in the &quot;formatting template&quot; field. This is an alternative argument for formattingTemplate and is useful if the &quot;formatting template&quot; field spans multiple lines.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the email notifier.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of your email notifier.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure. <strong>Also requires</strong> projectName</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project. <strong>Also requires</strong> the procedureName</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step. <strong>Also requires</strong> the procedureName and the projectName</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

### Positional arguments

**notifierName**

### Response

None or a status OK message.

### ec-perl

**syntax:** $cmdr->modifyEmailNotifier($notifierName, {<optionals>});

**Example**

```
$cmdr->modifyEmailNotifier("testNotifier",

    {eventType => "onCompletion",
```
```javascript
    projectName => "Project_test",
    procedureName => "Procedure_test",));

ectool

    syntax: ectool modifyEmailNotifier <notifierName> ...

Example

    ectool modifyEmailNotifier testNotifier
    --eventType onCompletion
    --projectName Project_test
    --procedureName Procedure_test

Back to Top

sendEmail

Facilitates sending an email from the command-line or a Command Job Step without setting up an Email Notifier.
This API is more dynamic than an email notifier because you do not need to setup some kind of a template beforehand. This API also makes sending email attachments easier than using a notifier template.

Instead of (or in addition to) specifying a configName, any of the configuration options for an email configuration can be specified as options. These options are: mailHost, mailPort, mailFrom, mailUser, and mailUserPassword.

Note: If both a configName and some or all of the configuration options are specified, the specified options override values stored in the configuration. In this case, the user must have both modify and execute permission on the configuration.

Specify the options you need to create the type of email message you want to send.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>configName</td>
<td>The name of the email configuration to use. If no configuration is specified, the configuration named &quot;default&quot; will be used. <strong>Note:</strong> The user must have &quot;execute&quot; permission on the configuration.</td>
</tr>
<tr>
<td>subject</td>
<td>The subject of the email message.</td>
</tr>
<tr>
<td>to</td>
<td>A &quot;To&quot; recipient for the email message. The recipient can be a user or group name or a complete email address. This option can be specified multiple times.</td>
</tr>
<tr>
<td>cc</td>
<td>A &quot;Cc&quot; recipient for the email message. The recipient can be a user or group name or a complete email address. This option can be specified multiple times.</td>
</tr>
<tr>
<td>bcc</td>
<td>A &quot;Bcc&quot; recipient for the email message. The recipient can be a user or group name or a complete email address. This option can be specified multiple times.</td>
</tr>
<tr>
<td>header</td>
<td>An RFC822 email header line (for example: &quot;reply-to: <a href="mailto:user@host.com">user@host.com</a>&quot;). This option can be specified multiple times.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>html</td>
<td>The body of a simple HTML message.</td>
</tr>
<tr>
<td>htmlFile</td>
<td>Reads the specified client-side file and uses it as the body of a simple HTML message.</td>
</tr>
<tr>
<td>text</td>
<td>The body of a simple text message.</td>
</tr>
<tr>
<td>textFile</td>
<td>Reads the specified client-side file and uses it as the body of a simple text message.</td>
</tr>
<tr>
<td>raw</td>
<td>A raw email message including headers to use as the basis for the email message. Additional options can be applied to this message. The value should be a properly formatted RFC822 message.</td>
</tr>
<tr>
<td>rawFile</td>
<td>Reads the specified client-side file and uses it as the entire mail message, including headers.</td>
</tr>
<tr>
<td>attachment</td>
<td>One or more client-side files to send as attachments. The filename extension is examined to determine the content-type. This option can be specified multiple times.</td>
</tr>
<tr>
<td>inline</td>
<td><code>&lt;contentId&gt;=&lt;fileName&gt; [&lt;contentId&gt;=&lt;fileName&gt; ...]</code> One or more inline attachments specified as a contentId and a client-side filename. The filename extension is examined to determine the content-type. The contentId can be referenced in an HTML body using the <code>cid:protocol</code>. For example: `&lt;img src=cid:myImage&quot; could reference --inline=myImage=image.jpg This option can be specified multiple times.</td>
</tr>
<tr>
<td>mailFrom</td>
<td>The &quot;From&quot; header to use when sending mail. Overrides the value from the email configuration if specified.</td>
</tr>
<tr>
<td>mailHost</td>
<td>The name of the mail server to use if no configName is specified. Overrides the value from the email configuration if specified.</td>
</tr>
<tr>
<td>mailPort</td>
<td>The mail server port to use if no configName is specified. Overrides the value from the email configuration if specified.</td>
</tr>
<tr>
<td>mailProtocol</td>
<td>The mail protocol. Must be either SMTP or SMTPS. Overrides the value from the email configuration if specified.</td>
</tr>
<tr>
<td>mailUser</td>
<td>The user account to use when authenticating to the mail server. Overrides the value from the email configuration if specified.</td>
</tr>
<tr>
<td>mailUserPassword</td>
<td>The password to use when authenticating to the mail server. Overrides the value from the email configuration if specified.</td>
</tr>
</tbody>
</table>
### multipartMode

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;none</td>
<td>mixed</td>
</tr>
<tr>
<td>none</td>
<td>non-multipart message</td>
</tr>
<tr>
<td>mixed</td>
<td>single-root multipart element of type &quot;mixed&quot;. Texts, inline elements, and attachments will be all be added to this root element.</td>
</tr>
<tr>
<td>related</td>
<td>multipart message with a single root multipart element of type &quot;related&quot;. Texts, inline elements, and attachments will be added to this root element. Works on most mail clients, except Lotus Notes.</td>
</tr>
<tr>
<td>mixedRelated</td>
<td>multipart element &quot;mixed&quot; plus a nested multipart element of type &quot;related&quot;. Texts and inline elements will be added to the nested &quot;related&quot; element, while attachments will be added to the &quot;mixed&quot; root element. Works on most mail clients other than Mac Mail and some situations on Outlook. If you experience problems, try &quot;related&quot;.</td>
</tr>
</tbody>
</table>

**Note:** multipartMode defaults to none unless there are multiple parts, in which case it defaults to mixedRelated. If both text and html arguments are specified, both values are sent as alternates in a multipart message.

### Positional arguments

None

### Response

None or status OK message.

### ec-perl

**syntax:** $cmdr->sendEmail

**Note:** The to, cc, bcc, header, and attachment options can have multiple values specified as an array. The inline option can have multiple values specified as an array of hashes with contentId and fileName values.

**Example**

```perl
$cmdr->sendEmail({
    configName => 'config1',
    subject => 'Test message',
    to => ['user1', 'user2'],
    html => '<html><body>Some stuff <img src=cid:image1/body/html>',
    inline => [{contentId => 'image1', fileName => 'image1.jpg'},
                {contentId => 'image2', fileName => 'image2.jpg'}],
    attachment => ['report1.html', 'report2.pdf']
})
```
**ectool**

**syntax:** `ectool sendEmail`

**Note:** Options that take multiple values may be specified as a single option with each value as a separate argument or as multiple options, each with a single argument.

**Examples**

```bash
ectool sendEmail
   --to user1 
   --to user2 
   --subject Test 
   --html '<html><body>Some stuff <img src="cid:image1"></body></html>' 
   --inline image1=image1.jpg 
   --inline image2=image2.jpg 
   --attachment report1.html 
   --attachment report2.pdf
```

```bash
ectool sendEmail
   --to user1 user2 
   --subject Test 
   --html '<html><body>Some stuff <img src="cid:image1"></body></html>' 
   --inline image1=image1.jpg image2=image2.jpg 
   --attachment report1.html report2.pdf
```

Back to Top
API Commands - Environment Requests

createEnvironment
createEnvironmentInventoryItem
deleteEnvironment
deleteEnvironmentInventoryItem
getEnvironment
getEnvironments
getEnvironmentApplications
getEnvironmentInventory
getEnvironmentInventoryItem
getEnvironmentInventoryItems
modifyEnvironment
modifyEnvironmentInventoryItem

createEnvironment

Creates a new environment.

**Required Arguments**

**projectName**

*Description:* Name for the project; must be unique among all projects.

*Argument Type:* String

**environmentName**

*Description:* Name of the environment; must be unique among all projects.

*Argument Type:* String

**Optional Arguments**

**applicationName**

*Description:* Create environment from the specified application; must be unique among all projects.

*Argument Type:* String

**applicationProjectName**

*Description:* Name of the application project.

*Argument Type:* String

**description**

*Description:* Comment text describing this object; not interpreted at all by the ElectricCommander platform.
**Argument Type:** String

environmentEnabled

**Description:** True to enable the environment.

**Argument Type:** Boolean

**Response**

Returns an environment element.

dc-perl

**Syntax:**

```perl
$<object>-&createEnvironment(<projectName>, <environmentName>,
  <optionals>);
```

**Example:**

```
$ec-&createEnvironment("Default", "aEnv", {environmentEnabled => "true",
  description => "aDescription"});
```

dc-tool

**Syntax:**

```bash
eckool createEnvironment <projectName> <environmentName> [optionals...]
```

**Example:**

```
eckool createEnvironment default newEnv --environmentEnabled true
  --description exampleText
```

**createEnvironmentInventoryItem**

Creates a new environment inventory item.

**Required Arguments**

**projectName**

**Description:** Name for the project; must be unique among all projects.

**Argument Type:** String

**environmentName**

**Description:** Name of the environment.

**Argument Type:** String

**applicationName**

**Description:** Name of the application that owns the inventory item.

**Argument Type:** String

**componentName**

**Description:** Component that owns the inventory item.
Argument Type: String
resourceName

Description: Resource where the item is installed.
Argument Type: String
artifactName

Description: Artifact name for the inventory item.
Argument Type: String
artifactVersion

Description: Artifact version for the inventory item.
Argument Type: String
Optional Arguments
artifactSource

Description: Source of the artifact.
Argument Type: String
artifactUrl

Description: URL of the artifact.
Argument Type: String
description

Description: Comment text describing this object; not interpreted by the ElectricCommander platform.
Argument Type: String
Response
Returns an environment inventory item.

ce-perl
Syntax:

$<object>->createEnvironmentInventoryItem(<projectName>, <environmentName>, <applicationName>, <componentName>, <resourceName>, <artifactName>, <artifactVersion>, {<optionals>});

Example:

$ec->createEnvironmentInventoryItem("Default", "aEnv", "App1", "ComponentA", "ResourceA", "Artifact1", "V3", {description => "aDescription"});

ectool
Syntax:

ectool createEnvironmentInventoryItem <projectName> <environmentName> <applicationName> <componentName> <resourceName> <artifactName> <artifactVersion> [optionals...]

Example:
ectool createEnvironmentInventoryItem Default aEnv Appl ComponentA ResourceA Artifact1 V3 --description aDescription

deleteEnvironment

Delete an environment.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

evironmentName

Description: Name of the environment; must be unique among all projects.

Argument Type: String

Optional Arguments

None

Response

None or a status OK message.

ec-perl

Syntax:

$<object>-deleteEnvironment(<projectName>, <environmentName>);

Example:

$cmdr->deleteEnvironment("Default", "envToDelete");

ectool

Syntax:

ectool deleteEnvironment <projectName> <environmentName>

Example:

ectool deleteEnvironment default envToDelete

deleteEnvironmentInventoryItem

Delete an inventory item.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

evironmentName
**Description**: Name of the environment.

**Argument Type**: String

(applicationName)

**Description**: Name of the application that owns the inventory item.

**Argument Type**: String

(componentName)

**Description**: Name of the component that owns the inventory item.

**Argument Type**: String

(resourceName)

**Description**: Name of the resource where the item is installed.

**Argument Type**: String

Optional Arguments

None

Response

None or a status OK message.

**ec-perl**

**Syntax:**

```
شخصيات>deleteEnvironmentInventoryItem(<projectName>, <environmentName>, <applicationName>, <componentName>, <resourceName>);
```

**Example:**

```
$cmdr->deleteEnvironmentInventoryItem("Default", "Env1A", "AppTest1", "Component1", "Server1");
```

**ectool**

**Syntax:**

```
ectool deleteEnvironmentInventoryItem <projectName> <environmentName> <applicationName> <componentName> <resourceName>
```

**Example:**

```
ectool deleteEnvironmentInventoryItem "Default" "Env1A" "AppTest1" "Component 1" "Server1"
```

**getEnvironment**

Retrieves an environment by name.

**Required Arguments**

(projectName)

**Description**: Name for the project; must be unique among all projects.
Argument Type: String

environmentName

Description: Name of the environment; must be unique among all projects.
Argument Type: String

Optional Arguments
None

Response
Retrieves an environment element.

ec-perl

Syntax:

$<object>-getEnvironment(<projectName>, <environmentName>);

Example:

$ec-getEnvironment("Default", "aEnv");

ectool

Syntax:

ec tool getEnvironment <projectName> <environmentName>

Example:

ectool getEnvironment default newEnv

getEnvironments

Retrieves all environments in a project.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.
Argument Type: String

Optional Arguments
None

Response
Retrieves zero or more environment elements.

ec-perl

Syntax:

$<object>-getEnvironments(<projectName>);

Example:

$ec-getEnvironments("Default");
Electrical Commander

**ectool**

**Syntax:**

```bash
ectool getEnvironments <projectName>
```

**Example:**

```bash
ectool getEnvironments default
```

---

**getEnvironmentApplications**

Retrieves a list of applications installed on the given environment.

**projectName**

**Description:** Name for the project; must be unique among all projects.

**Argument Type:** String

**environmentName**

**Description:** Name of the environment.

**Argument Type:** String

**Optional Arguments**

None

**Response**

Retrieves a list of applications for the specified environment.

**ec-perl**

**Syntax:**

```perl
$<object>->getEnvironmentApplications(<projectName>, <environmentName>);
```

**Example:**

```perl
$ec->getEnvironmentApplications("Default", "aEnv");
```

**ectool**

**Syntax:**

```bash
ectool getEnvironmentApplications <projectName> <environmentName>
```

**Example:**

```bash
ectool getEnvironmentApplications default newEnv
```

---

**getEnvironmentInventory**

Retrieves a per-component grouped list of inventory items.

**Required Arguments**

**projectName**

**Description:** Name for the project; must be unique among all projects.
getEnvironmentInventoryItem

Retrieves an inventory item.

**Required Arguments**

projectName

- **Description:** Name for the project; must be unique among all projects.
- **Argument Type:** String

environmentName

- **Description:** Name of the environment.
- **Argument Type:** String

applicationName

- **Description:** Name of the application that owns the inventory item.
- **Argument Type:** String
**Argument Type:** String

compartmentName

**Description:** Name of the component that owns the inventory item.

**Argument Type:** String

resourceName

**Description:** Name of the resource where the item is installed.

**Argument Type:** String

Optional Arguments

None

**Response**

Retrieves an inventory item.

**ec-perl**

**Syntax:**

```perl
$<object>-getEnvironmentInventoryItem(<projectName>, <environmentName>, <applicationName>, <componentName>, <resourceName>);
```

**Example:**

```perl
$ec->getEnvironmentInventoryItem("Default", "aEnv", "App1", "Component1", "Server1");
```

**ectool**

**Syntax:**

```bash
ectool getEnvironmentInventoryItem <projectName> <environmentName> <applicationName> <componentName> <resourceName>
```

**Example:**

```bash
ectool getEnvironmentInventoryItem default newEnv App1 Component1 Server1
```

**getEnvironmentInventoryItems**

Retrieves all inventory items for a given environment.

**Required Arguments**

**projectName**

**Description:** Name for the project; must be unique among all projects.

**Argument Type:** String

**environmentName**

**Description:** Name of the environment.

**Argument Type:** String
Optional Arguments
None

Response
Retrieves all inventory items for the specified environment.

ec-perl
Syntax:
`$<object>-getEnvironmentInventoryItems(<projectName>, <environmentName>);`

Example:
`$ec->getEnvironmentInventoryItems("Default", "aEnv");`

ectool
Syntax:
`ectool getEnvironmentInventoryItems <projectName> <environmentName>`

Example:
`ectool getEnvironmentInventoryItems default newEnv`

modifyEnvironment
Modifies an environment.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

environmentName

Description: Name of the environment; must be unique among all projects.

Argument Type: String

Optional Arguments
description

Description: Comment text describing this object; not interpreted at all by the ElectricCommander platform.

Argument Type: String

evironmentEnabled

Description: True to enable the environment.

Argument Type: Boolean

newName

Description: New name for an existing object that is being renamed.
**Argument Type:** String

**Response**
Retrieves an updated environment element.

**ec-perl**

**Syntax:**
```perl
$<object>-modifyEnvironment(<projectName>, <environmentName>,
    {<optionals>});
```

**Example:**
```perl
$ec-modifyEnvironment("Default", "aEnv", {newName => "upDatedName",
    description => "aNewDescription"});
```

**ectool**

**Syntax:**
```bash
ectool modifyEnvironment <projectName> <environmentName> 
    [optionals...]
```

**Example:**
```bash
ectool modifyEnvironment default testEnv --newName modEnv
    --description exampleText
```

**modifyEnvironmentInventoryItem**
Modifies an existing environment inventory item.

**Required Arguments**

**projectName**

- **Description:** Name for the project; must be unique among all projects.
- **Argument Type:** String

**environmentName**

- **Description:** Name of the environment.
- **Argument Type:** String

**applicationName**

- **Description:** Name of the application that owns the inventory item.
- **Argument Type:** String

**componentName**

- **Description:** Name of the component that owns the inventory item.
- **Argument Type:** String

**resourceName**

- **Description:** Name of the resource where the item is installed.
Argument Type: String
artifactName
Description: Name of the artifact for the inventory item.
Argument Type: String
artifactVersion
Description: Version of the artifact for the inventory item.
Argument Type: String

Optional Arguments
artifactSource
Description: Source of the artifact.
Argument Type: String
artifactUrl
Description: URL of the artifact.
Argument Type: String
description
Description: Comment text describing this object; not interpreted by the ElectricCommander platform.
Argument Type: String

Response
Retrieves an updated environment inventory item.

e-cperl Syntax:
$<object>-modifyEnvironmentInventoryItem(<projectName>, <environmentName>,
<applicationName>, <componentName>, <resourceName>, <artifactName>,
<artifactVersion> [<optionals>]);

Example:
$ec-modifyEnvironmentInventoryItem("Default", "aEnv", "Appl", "Component1",
"Server1", "Artifact1", "V3");

ectool Syntax:
ectool modifyEnvironmentInventoryItem <projectName> <environmentName>
<applicationName> <componentName> <resourceName> <artifactName>
<artifactVersion> [optionals...]

Example:
ectool modifyEnvironmentInventoryItem default testEnv Appl Component1 Server1
Artifact1 V3
API Commands - Environment Tier

createEnvironmentTier

Creates a new environment tier.

**Required Arguments**

projectName

*Description:* Name for the project; must be unique among all projects.

*Argument Type:* String

environmentName

*Description:* Name of the environment which must be unique among all environments for the project; must be unique among all projects.

*Argument Type:* String

environmentTierName

*Description:* Name of the environment tier; must be unique among all tiers for the environment.

*Argument Type:* String

**Optional Arguments**

description

*Description:* Comment text describing this object; not interpreted at all by the ElectricCommander platform.

*Argument Type:* String

Response

Returns an environment tier element.

ec-perl

**Syntax:**

```perl
$<object>-createEnvironmentTier(<projectName>, <environmentName>,
<environmentTierName>, {<optionals>});
```

**Example:**

```perl
$ec->createEnvironmentTier("Default", "newEnv", "envTier2",
{description => "Description"});
```
ectool
Syntax:

    ectool createEnvironmentTier <projectName> <environmentName> <environmentTierName> [optionals...]

Example:

    ectool createEnvironmentTier default newEnv envTier1 --description exampleText

deleteEnvironmentTier
Deletes an environment tier.

Required Arguments

projectName

    Description: Name of the project; must be unique among all projects.
    Argument Type: String

environmentName

    Description: Name of the environment that must be unique among all environments for the project; must be unique among all projects.
    Argument Type: String

environmentTierName

    Description: Name of the environment tier; must be unique among all tiers for the environment.
    Argument Type: String

Optional Arguments

None

Response

None or a status OK message.

ec-perl
Syntax:

    $<object>-&deleteEnvironmentTier(<projectName>, <environmentName>, <environmentTierName>);

Example:

    $ec-&deleteEnvironmentTier("Default", "newEnv", "tierToDelete");

ectool
Syntax:

    ectool deleteEnvironmentTier <projectName> <environmentName> <environmentTierName>

Example:
getEnvironmentTier

Retrieves an environment tier by name.

**Required Arguments**

- projectName
  - **Description:** Name of the project; must be unique among all projects.
  - **Argument Type:** String

- environmentName
  - **Description:** Name of the environment which must be unique among all environments for the project; must be unique among all projects.
  - **Argument Type:** String

- environmentTierName
  - **Description:** Name of the environment tier; must be unique among all tiers for the environment.
  - **Argument Type:** String

**Optional Arguments**

- None

**Response**

Retrieves an environment tier element.

**ec-perl**

**Syntax:**

```perl
$result = $<object>-getEnvironmentTier(<projectName>, <environmentName>, <environmentTierName>);
```

**Example:**

```perl
$ec->getEnvironmentTier("Default", "newEnv", "envTier2");
```

**ectool**

**Syntax:**

```bash
ectool getEnvironmentTier <projectName> <environmentName> <environmentTierName>
```

**Example:**

```bash
ectool getEnvironmentTier default newEnv envTier1
```

getEnvironmentTiers

Retrieves all environment tiers in an environment.
Required Arguments

**projectName**
- **Description:** Name of the project; must be unique among all projects.
- **Argument Type:** String

**environmentName**
- **Description:** Name of the environment that must be unique among all environments for the project; must be unique among all projects.
- **Argument Type:** String

Optional Arguments
- None

Response
- Retrieves zero or more environment tier elements.

**ec-perl**
- **Syntax:**
  ```perl
  $<object>-getEnvironmentTiers(<projectName>, <environmentName>);
  ```
- **Example:**
  ```perl
  $ec->getEnvironmentTiers("Default", "newEnv");
  ```

**ectool**
- **Syntax:**
  ```bash
  ectool getEnvironmentTiers <projectName> <environmentName>
  ```
- **Example:**
  ```bash
  ectool getEnvironmentTiers default newEnv
  ```

**modifyEnvironmentTier**

Modifies an environment tier.

Required Arguments

**projectName**
- **Description:** Name of the project; must be unique among all projects.
- **Argument Type:** String

**environmentName**
- **Description:** Name of the environment which must be unique among all environments for the project; must be unique among all projects.
- **Argument Type:** String

**environmentTierName**
- **Description:** Name of the environment tier; must be unique among all tiers for the environment.
**Argument Type:** String

**Optional Arguments**

description

**Description:** Comment text describing this object; not interpreted at all by the ElectricCommander platform.

**Argument Type:** String

newName

**Description:** New name for an existing object that is being renamed.

**Argument Type:** String

**Response**

Retrieves an updated environment tier element.

**ece-perl**

Syntax:

```perl
$<object>->modifyEnvironmentTier(<projectName>, <environmentName>, <environmentTierName>, {<optionals>});
```

Example:

```perl
$ec->modifyEnvironmentTier("Default", "newEnv", "envTier2", 
    {newName => "envTierB", description => "New_Description"});
```

**ectool**

Syntax:

```bash
ectool modifyEnvironmentTier <projectName> <environmentName> <environmentTierName> [optionals...]
```

Example:

```bash
ectool modifyEnvironmentTier default newEnv envTier1 --description new_exampleText --newName envTierA
```
# API Commands - Gateways/Zones Management

**createGateway**

Creates a new gateway.

**Scenario:** You have two zones, ZoneA and ZoneB. ResourceA in ZoneA is accessible from ResourceB in ZoneB, and conversely—communication between specified gateway resources is enabled with host/port information recorded in each resource object. Other resources in each zone are restricted to talking to resources within their zone only. Creating a gateway between ResourceA and ResourceB to link the two zones enables resources from one zone to communicate with the other using ResourceA and ResourceB.

You must specify `gatewayName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>gatewayDisabled</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway. Supply any of your choice, but the name must be unique among other gateway names.</td>
</tr>
<tr>
<td>hostName1</td>
<td>The agent host name where Resource1 resides. This host name is used by Resource2 to communicate with Resource1. Do not specify this option if you want to use the host name from Resource1's definition.</td>
</tr>
<tr>
<td>hostName2</td>
<td>The agent host name where Resource2 resides. This host name is used by Resource1 to communicate with Resource2. Do not specify this option if you want to use the host name from Resource2's definition.</td>
</tr>
<tr>
<td>port1</td>
<td>The port number used by Resource1 - defaults to the port number used by the resource.</td>
</tr>
<tr>
<td>port2</td>
<td>The port number used by Resource2 - defaults to the port number used by the resource.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName1</td>
<td>The name of your choice for the first of two required gateway resources. Do <strong>not</strong> include &quot;spaces&quot; in a resource name.</td>
</tr>
<tr>
<td>resourceName2</td>
<td>The name of your choice for the second of two required gateway resources. Do <strong>not</strong> include &quot;spaces&quot; in a resource name.</td>
</tr>
</tbody>
</table>

#### Positional arguments

- **gatewayName**

#### Response

- **Returns** a gateway object.

#### ec-perl

- **syntax:**
  ```perl
  $cmdr->createGateway (<gatewayName>, {<optionals>});
  ```

- **Example**
  ```perl
  $cmdr->createGateway ("AB_Gateway",
  {description => "Gateway linking ZoneA and ZoneB",
   resourceName1 => "ResourceA",
   resourceName2 => "ResourceB")};
  ```

#### ectool

- **syntax:**
  ```bash
  ectool createGateway <gatewayName> ...
  ```

- **Example**
  ```bash
  ectool createGateway AB_Gateway --description "Gateway linking ZoneA and ZoneB"
  --resourceName1 "ResourceA"
  --resourceName2 "ResourceB"
  ```

---

**deleteGateway**

Deletes a gateway. You must supply a **gatewayName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayName</td>
<td>The name of the gateway to delete.</td>
</tr>
</tbody>
</table>

#### Positional arguments

- **gatewayName**
**Response**
None

**ec-perl**

* syntax: $cmdr->deleteGateway (<gatewayName>);

* Example

$cmdr->deleteGateway("AB_Gateway");

**ectool**

* syntax: ectool deleteGateway <gatewayName>

* Example

ectool deleteGateway "AB_Gateway"

---

**getGateway**

Finds a gateway by name.

You must specify a *gatewayName*.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayName</td>
<td>The name of the gateway you want to find.</td>
</tr>
</tbody>
</table>

**Positional arguments**

gatewayName

**Response**

Returns one *gateway* element.

**ec-perl**

* syntax: $cmdr->getGateway (<gatewayName>);

* Example

$cmdr->getGateway("AB_Gateway");

**ectool**

* syntax: ectool getGateway <gatewayName>

* Example

ectool getGateway AB_Gateway
getGateways

Retrieves all gateways.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

Positional arguments
None.

Response
Returns one or more gateway elements.

ecc-perl
	syntax: $cmdr->getGateways();

Example
$cmdr->getGateways();

ectool
	syntax: ectool getGateways

Example
ectool getGateways

modifyGateway

Modifies an existing gateway.

You must specify a gatewayName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must</td>
</tr>
<tr>
<td></td>
<td>surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed</td>
</tr>
<tr>
<td></td>
<td>in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>gatewayDisabled</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway you want to modify.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostName1</td>
<td>The agent host name where Resource1 resides. This host name is used by Resource2 to communicate with Resource1. Do not specify this option if you want to use the host name from Resource1's definition.</td>
</tr>
<tr>
<td>hostName2</td>
<td>The agent host name where Resource2 resides. This host name is used by Resource1 to communicate with Resource2. Do not specify this option if you want to use the host name from Resource2's definition.</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the gateway.</td>
</tr>
<tr>
<td>port1</td>
<td>The port number used by Resource1 - defaults to the port number used by the resource.</td>
</tr>
<tr>
<td>port2</td>
<td>The port number used by Resource2 - defaults to the port number used by the resource.</td>
</tr>
<tr>
<td>resourceName1</td>
<td>The name of your choice for the first of two required gateway resources. Do not include &quot;spaces&quot; in a resource name.</td>
</tr>
<tr>
<td>resourceName2</td>
<td>The name of your choice for the second of two required gateway resources. Do not include &quot;spaces&quot; in a resource name.</td>
</tr>
</tbody>
</table>

### Positional arguments

gatewayName

### Response

An updated gateway object.

#### ec-perl

**syntax:** `$cmdr->modifyGateway (<gatewayName>, {...});`

**Example**

```
$cmdr->modifyGateway("AB_Gateway",
    {description=> "Gateway linking zoneA and zoneB",
     resourceName1=> "ResourceA",
     resourceName2=> "ResourceB"});
```

#### ectool

**syntax:** `ectool modifyGateway <gatewayName> ...`

**Example**

```
ectool modifyGateway AB_Gateway --description "Gateway linking ZoneA and ZoneB"
   --resourceName1 "ResourceA"
   --resourceName2 "ResourceB"
```
**createZone**

Creates a new zone.

You must specify a `zoneName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>description</strong></td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td><strong>zoneName</strong></td>
<td>The name of the zone. Supply any unique name of your choice.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`zoneName`

**Response**

Returns a zone object.

**ec-perl**

`syntax`: `$cmdr->createZone (<zoneName>, { ... });`

**Example**

```perl
$cmdr->createZone("DevZone", { description => "Zone containing resources that the dev group uses." });
```

**ectool**

`syntax`: `ectool createZone <zoneName> ...

**Example**

`ectool createZone DevZone --description "Zone containing resources that the dev group uses."`

**deleteZone**

Deletes an existing zone.

You must specify a `zoneName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>zoneName</strong></td>
<td>The name of the zone to delete.</td>
</tr>
</tbody>
</table>
**Positional arguments**

*zoneName*

**Response**

None

**ec-perl**

*syntax:* `$cmdr->deleteZone (<zoneName>);`

*Example*

`$cmdr->deleteZone ("DevZone");`

**ectool**

*syntax:* `ectool deleteZone <zoneName>`

*Example*

`ectool deleteZone DevZone`

---

**getZone**

Finds a zone by name.

You must specify a *zoneName*.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoneName</td>
<td>The name of the zone you want to find.</td>
</tr>
</tbody>
</table>

**Positional arguments**

*zoneName*

**Response**

Returns a *zone* element, including a list of resources belonging to the zone.

**ec-perl**

*syntax:* `$cmdr->getZone (<zoneName>);`

*Example*

`$cmdr->getZone ("DevZone");`

**ectool**

*syntax:* `ectool getZone <zoneName>`
**getZones**

Retrieves all zones.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Returns a zone object.

**ec-perl**

*Syntax:* $cmdr->getZones();

*Example*

$cmdr->getZones();

**ectool**

*Syntax:* ectool getZones

*Example*

ectool getZones

**modifyZone**

Modifies an existing zone.

You must specify a `zoneName`. 
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td>newName</td>
<td>Supply any unique name of your choice to rename the zone.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of this zone.</td>
</tr>
</tbody>
</table>

**Positional arguments**

zoneName

**Response**

Returns an updated zone element.

**ec-perl**

*Syntax:* `cmdr->modifyZone (<zoneName>, {...});`

*Example*

`cmdr->modifyZone ("DevZone", {description => "Zone containing resources that the dev group uses."});`

**ectool**

*Syntax:* `ectool modifyZone <zoneName> ...`

*Example*

`ectool modifyZone DevZone --description "Zone containing resources that the dev group uses."`

[Back to Top]
# API Commands - Job Management

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortAllJobs</td>
<td>Aborts all running jobs.</td>
</tr>
<tr>
<td>abortJob</td>
<td></td>
</tr>
<tr>
<td>abortJobStep</td>
<td></td>
</tr>
<tr>
<td>deleteJob</td>
<td></td>
</tr>
<tr>
<td>findJobSteps</td>
<td></td>
</tr>
<tr>
<td>getJobDetails</td>
<td></td>
</tr>
<tr>
<td>getJobInfo</td>
<td></td>
</tr>
<tr>
<td>getJobNotes</td>
<td></td>
</tr>
<tr>
<td>getJobs</td>
<td></td>
</tr>
<tr>
<td>getJobsForSchedule</td>
<td></td>
</tr>
<tr>
<td>getJobStatus</td>
<td></td>
</tr>
<tr>
<td>getJobStepDetails</td>
<td></td>
</tr>
<tr>
<td>getJobStepStatus</td>
<td></td>
</tr>
<tr>
<td>moveJobs</td>
<td></td>
</tr>
<tr>
<td>runProcedure</td>
<td></td>
</tr>
<tr>
<td>setJobName</td>
<td></td>
</tr>
<tr>
<td>getJobsForSchedule</td>
<td></td>
</tr>
<tr>
<td>getJobStatus</td>
<td></td>
</tr>
<tr>
<td>getJobStepDetails</td>
<td></td>
</tr>
<tr>
<td>getJobStepStatus</td>
<td></td>
</tr>
<tr>
<td>moveJobs</td>
<td></td>
</tr>
<tr>
<td>runProcedure</td>
<td></td>
</tr>
<tr>
<td>setJobName</td>
<td></td>
</tr>
</tbody>
</table>

## External Job APIs

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>completeJob</td>
<td>modifyJob</td>
</tr>
<tr>
<td>completeJobStep</td>
<td>modifyJobStep</td>
</tr>
<tr>
<td>createJob</td>
<td></td>
</tr>
<tr>
<td>createJobStep</td>
<td></td>
</tr>
</tbody>
</table>

### abortAllJobs

Aborts all running jobs.

#### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>reason</td>
<td>A string added to the aborted job/jobstep that describes or records the reason for the abort. The server records this value, but places no meaning on the string - similar to a text Description &quot;for your reference only.&quot;</td>
</tr>
</tbody>
</table>

#### Positional arguments

None

#### Response

None or status OK message.

#### ec-perl

**Syntax:**

```perl
$cmdr->abortAllJobs({...});
```

**Example**

```perl
$cmdr->abortAllJobs({force => 1});
```
ectool

**syntax:**  ectool abortAllJobs ...

**Example**

```
ectool abortAllJobs --force 1
```

AbortJob

Aborts a running job.

You must supply a jobId.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>reason</td>
<td>A string added to the aborted job/jobstep that describes or records the reason for the abort. The server records this value, but places no meaning on the string - similar to a text Description &quot;for your reference only.&quot;</td>
</tr>
</tbody>
</table>

**Positional arguments**

jobId

**Response**

None or status OK message.

ec-perl

**syntax:**  $cmdr->abortJob(<jobId>, {...});

**Example**

```
$cmdr->abortJob(4fa765dd-73f1-11e3-b67e-b0a420524153, {force => 1});
```

ectool

**syntax:**  ectool abortJob <jobId> ...

**Example**

```
ectool abortJob 4fa765dd-73f1-11e3-b67e-b0a420524153 --force 1
```
The `abortJobStep` command aborts any type of step—command step or subprocedure step.

Aborting a subprocedure step aborts all steps of the subprocedure as well. Steps marked "always run" will still run to completion unless the "force" flag is specified.

You must specify a `jobStepId`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The <code>jobStep</code> to abort - the unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>reason</td>
<td>A string added to the aborted <code>job/jobstep</code> that describes or records the reason for the abort. The server records this value, but places no meaning on the string - similar to a text Description &quot;for your reference only.&quot;</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `jobStepId`

**Response**

None or status OK message.

**ec-perl**

- **syntax**: `$cmdr->abortJobStep(<jobStepId>, {...});`

  - **Example**: `$cmdr->abortJobStep(5da765dd-73f1-11e3-b67e-b0a420524153, {force => 1});`

**ectool**

- **syntax**: `ectool abortJobStep <jobStepId> ...

  - **Example**: `ectool abortJobStep 5da765dd-73f1-11e3-b67e-b0a420524153`

**deleteJob**

The `deleteJob` command deletes a job from the ElectricCommander database.

You must specify a `jobId`.
### API Commands - Job Management

#### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
</tbody>
</table>

#### Positional arguments

**jobId**

#### Response

None or a status OK message.

**ec-perl**

- **syntax**: `$cmdr->deleteJob(<jobId>);`

- **Example**
  ```
  $cmdr->deleteJob(4fa765dd-73f1-11e3-b67e-b0a420524153);
  ```

**ectool**

- **syntax**: `ectool deleteJob <jobId>`

- **Example**
  ```
  ectool deleteJob 4fa765dd-73f1-11e3-b67e-b0a420524153
  ```

#### Back to Top

### findJobSteps

Returns a list of job steps from a single job or from a single subprocedure job step. This API is used by the Job Details web page in the Commander UI. The elements in the list are returned in their natural "job order".

You must specify either a **jobId** or a **jobStepId**, but not both.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique identifier for the job whose steps you want to retrieve - assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step whose job steps you want to retrieve - assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>filter</td>
<td>A list of zero or more filter criteria definitions used to define objects to find. See the <strong>findObjects</strong> API for complete description for using filters.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>numObjects</td>
<td>&lt;full object count&gt; This specifies the number of full job steps (not just the IDs) returned in the response. Returned job steps will be from the beginning of the list. If numObjects is not specified, all job steps in the list of object IDs are returned. Any and all job steps can be retrieved using the getObjects command.</td>
</tr>
<tr>
<td>select</td>
<td>This is an unordered list of property names that specify additional top-level properties to return for each object. See the code example for findObjects for instructions on forming the list and passing it to the ElectricCommander Perl API.</td>
</tr>
</tbody>
</table>

### Positional arguments

jobId, or jobStepId

### Response

One or more jobStep elements.

### ec-perl

*Syntax:* $cmdr->findJobSteps({<optionals>});

**Example 1**

```perl
my $xPath = $cmdr->findJobSteps({
    jobId => "4fa765dd-73f1-11e3-b67e-b0a420524153",
    select => [{propertyName => 'charEncoding'}, {propertyName => 'abc'}])

print "Return data from Commander:\n" .
    $xPath->findnodes_as_string("/"). "\n";
```

**Example 2**

```perl
my $xPath = $cmdr->findJobSteps({jobStepId => "5da765dd-73f1-11e3-b67e-b0a420524153")

print "Return data from Commander:\n" .
    $xPath->findnodes_as_string("/"). "\n";
```

### ectool

Not supported.

### getJobDetails

Retrieves complete information about a job, including details from each job step.

You must specify a jobId.
Arguments | Descriptions
---|---
jobId | The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.
structureOnly | `<Boolean flag - 0|1|true|false>` Reduces the amount of information returned to minimal structural information.

Positional arguments
jobId

Response
One `job` element, including one or more `jobStep` elements.

c-error
**syntax:**
```
$cmdr->getJobDetails(<jobId>, {<optionals>});
```

**Example**
```
$cmdr->getJobDetails(4fa765dd-73f1-11e3-b67e-b0a420524153, {structureOnly => 1});
```

ectool
**syntax:**
ectool getJobDetails <jobId> ...

**Example**
ectool getJobDetails 4fa765dd-73f1-11e3-b67e-b0a420524153 --structureOnly 1

getJobInfo
Retrieves all information about a job, except job step information.
You must specify a `jobId`.

Arguments | Descriptions
---|---
jobId | The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.

Positional arguments
jobId

Response
One `job` element.
**getJobInfo**

Retrieves the job information from a job.

### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
</tbody>
</table>

### ec-perl Syntax

```perl
$cmdr->getJobInfo(<jobId>);
```

**Example**

```perl
$cmdr->getJobInfo(4fa765dd-73f1-11e3-b67e-b0a420524153);
```

### ectool Syntax

```bash
ectool getJobInfo <jobId>
```

**Example**

```bash
ectool getJobInfo 4fa765dd-73f1-11e3-b67e-b0a420524153
```

---

**getJobNotes**

Retrieves the notes property sheet from a job.

You must specify a `jobId`.

### Positional arguments

`jobId`

**Response**

A `propertySheet` element that contains the job.

### ec-perl Syntax

```perl
$cmdr->getJobNotes(<jobId>);
```

**Example**

```perl
$cmdr->getJobNotes(4fa765dd-73f1-11e3-b67e-b0a420524153);
```

### ectool Syntax

```bash
ectool getJobNotes <jobId>
```

**Example**

```bash
ectool getJobNotes 4fa765dd-73f1-11e3-b67e-b0a420524153
```
getJobs

Retrieves summary information for a list of jobs. By default, all jobs are returned.

Notes:
1. If using sortKey or sortOrder, you must use both arguments together.
2. You can use firstResult and maxResults together or separately to select a limited sub-list of jobs for the result set.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstResult</td>
<td>&lt;index number&gt; 0-based index identifies the first element returned from filtered, sorted result set.</td>
</tr>
<tr>
<td>maxResults</td>
<td>&lt;result count&gt; This number sets the maximum number of returned jobs.</td>
</tr>
<tr>
<td>sortKey</td>
<td>&lt;jobId</td>
</tr>
<tr>
<td>sortOrder</td>
<td>&lt;ascending</td>
</tr>
<tr>
<td>status</td>
<td>&lt;running</td>
</tr>
</tbody>
</table>

Positional arguments

None

Response

One or more job elements. A job element contains summary information only.

ec-perl

Syntax: $cmdr->getJobs({...});

Examples

How do I get the first 10 jobs (index 0-9)?

$cmdr->getJobs({maxResults=>10});

How do I get the next 10 jobs (index 10-19)?

$cmdr->getJobs({firstResult=>10, maxResults =>10});

How do I get the most recent job by start time?

$cmdr->getJobs({sortKey=>'start', sortOrder=>'descending', maxResults =>1});

ectool

Syntax: ectool getJobs ...
**Examples**

How do I get the first 10 jobs (index 0-9)?

```bash
ectool getJobs --maxResults 10
```

How do I get the next 10 jobs (index 10-19)?

```bash
ectool getJobs --firstResult 10 --maxResults 10
```

How do I get the most recent job by start time?

```bash
ectool getJobs --sortKey start --sortOrder descending --maxResults 1
```

---

**getJobsForSchedule**

Retrieves jobs started by a specific schedule.

You must specify a **projectName** and **scheduleName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that contains this schedule.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule that launched these jobs.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, scheduleName`

**Response**

Returns an XML stream containing any number of **job** elements. The **job** elements contain summary information only.

**ec-perl**

**Syntax:**

```perl
$cmdr->getJobsForSchedule(<projectName>, <scheduleName>);
```

**Example**

```perl
$cmdr->getJobsForSchedule('Test', 'ea1');
```

**ectool**

**Syntax:**

```bash
ectool getJobsForSchedule <projectName> <scheduleName>
```

**Example**

```bash
ectool getJobsForSchedule Test ea1
```

---

Back to Top
**getJobStatus**

Retrieves the status of a job.

You must specify the `jobId`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>jobId</code></td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`jobId`

**Response**

Values for `status` and `outcome` as follows:

Possible values for `status`:

- **pending** - The job is not yet runnable—it is waiting for other steps to complete first. A job should not stay in this state for longer than a few seconds.
- **runnable** - The job is ready to run, but it is waiting for a resource to become available.
- **running** - The job is assigned to a resource and is executing the step command.
- **completed** - The job finished executing.

Possible values for `outcome`: The outcome is accurate only if the job status is "completed."

- **success** - The job finished successfully.
- **warning** - The job completed with no errors, but encountered some suspicious conditions.
- **error** - The job has finished execution with errors.

**ec-perl**

*Syntax:* $cmdr->getJobStatus(<jobId>);

*Example*

```
$cmdr->getJobStatus(4fa765dd-73f1-11e3-b67e-b0a420524153);
```

**ectool**

*Syntax:* ectool getJobStatus <jobId>

*Example*

```
ectool getJobStatus 4fa765dd-73f1-11e3-b67e-b0a420524153
```

**getJobStepDetails**

Retrieves details for a job step.
You may never need to use this command. This information is available for all job steps in a job by using the `getJobDetails` command. The `getJobStepDetails` command can be used to refresh data for a single step if you need an update in real time.

You must specify `jobStepId`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>structureOnly</td>
<td>`&lt;Boolean flag - &lt;0</td>
</tr>
</tbody>
</table>

**Positional arguments**

`jobStepId`

**Response**

A `jobStep` element.

**ec-perl**

*Syntax:* `$cmdr->getJobStepDetails(<jobStepId>, {...});`

*Example*

$cmdr->getJobStepDetails(5da765dd-73f1-11e3-b67e-b0a420524153);

**ectool**

*Syntax:* `ectool getJobStepDetails <jobStepId> ...

*Example*

ectool getJobStepDetails 5da765dd-73f1-11e3-b67e-b0a420524153

**getJobStepStatus**

Retrieves the status of a job step.

You must specify the `jobStepId`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`jobStepId`
Response
A status tag - for example: <status>completed</status>

Possible values for status:
  pending - The job step is not yet runnable--it is waiting for other steps to complete first. A job should not stay in this state for longer than a few seconds.
  runnable - The job step is ready to run, but it is waiting for a resource to become available.
  running - The job step is assigned to a resource and is executing the step command.
  completed - The job step finished executing.

ec-perl
  syntax: $cmdr->getJobStepStatus(<jobStepId>, {..});

Example
$cmdr->getJobStepStatus(5da765dd-73f1-11e3-b67e-b0a420524153);

ectool
  syntax: ectool getJobStepStatus <jobStepId>

Example
ectool getJobStepStatus 5da765dd-73f1-11e3-b67e-b0a420524153

moveJobs
Moves jobs from one project to another project.
You must specify sourceProject and destinationProject.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceProject</td>
<td>The name of the project that contains the jobs you want to move.</td>
</tr>
<tr>
<td>destinationProject</td>
<td>The new project that will contain the jobs.</td>
</tr>
</tbody>
</table>

Positional arguments
  sourceProject, destinationProject

Response
None or a status OK message.

ec-perl
  syntax: $cmdr->moveJobs(<sourceProject>, <destinationProject>);
**Example**

```
$cmdr->moveJobs("ProjectA", "ProjectB");
```

**ectool**

**syntax:** `ectool moveJobs <sourceProject> <destinationProject> ...

**Example**

```
ectool moveJobs "ProjectA" "ProjectB"
```

---

**runProcedure**

Creates and starts a new job using a procedure directly or a procedure specified indirectly through a schedule. Returns a new job ID. If the `pollInterval` option is provided, wait until the job completes up to a maximum of `timeout` seconds (if also provided). If the `scheduleName` option is provided, the parameters provided by that schedule will be used.

- **runProcedure credentials** - two types of credentials can be passed to `runProcedure`:
  - Impersonation credentials
  - Credential parameters

**Impersonation credentials**

Impersonation credentials are used to set the top level impersonation credential for a job. If specified, the impersonation credential [on the job] is used as the default impersonation credential for all steps in the job.

The impersonation credential can be specified in two ways. If the `credentialName` argument is supplied, the job looks for the named credential specified. If the user has execute permission on the specified credential, `runProcedure` is allowed to start the job.

If the `userName` and `password` arguments are supplied, the job creates a transient credential to contain the pair. The transient credential is used by the job and then discarded when the job completes.

Only one of `credentialName` or `userName` should be specified. If both are specified, only `userName` is used.

Neither can be specified if the procedure being run already has a credential defined on the procedure or the project.

**Credential parameters**

If the procedure defines one or more credential parameters, `runProcedure` must specify a credential to use for each parameter. The `actualParameter` argument identifies the credential name to use for the parameter, and the `credential` argument specifies the user name for each defined credential. For each credential specified, ectool prompts for a password.

For example, for a procedure named `proc1` with a single credential parameter named `param1`. The following command could be used to pass a transient credential where the user name is `joe` and the password is `plumber`:

```
$ ectool runProcedure test --procedureName proc1 \
   --actualParameter param1=cred1 --credential cred1=joe \
   cred1 password: plumber
```
Multiple parameters or credentials can be specified by having additional *name=value* pairs after the `actualParameter` or `credential` arguments. The same credential can be specified as the value for more than one actual parameter.

You must specify a `projectName` and either a `procedureName` or a `scheduleName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure. Used in conjunction with <code>procedureName</code> to set the value of the actual parameters. Do not use this argument with <code>scheduleName</code>.</td>
</tr>
<tr>
<td>credential</td>
<td>Use the following syntax to specify a credential: <code>&lt;credName&gt;=&lt;userName&gt; [&lt;credName&gt;=&lt;userName&gt; ...]</code></td>
</tr>
</tbody>
</table>
| credentialName  | `credentialName` can be one of two forms:  
|                 | **relative**  
|                 | (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.  
|                 | **absolute**  
|                 | (for example, "projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project. |
| destinationProject | This argument is used to specify the name of the destination project.                                                                 |
| password        | The password for the user running the procedure.                                                                                           |
| priority        | Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number.  
|                 | **Priority values are:** low|normal|high|highest                                                                                           |
| procedureName   | The name of the procedure you want to run.                                                                                                  |
| projectName     | The name of the project that contains the procedure you want to run.                                                                        |
| scheduleName    | The name of the schedule. Use this option if you want to use the parameters from an existing specific schedule.                              |
| userName        | The name of the user who is running the procedure.                                                                                          |
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Note:</strong></td>
<td>The following two arguments are used to control whether <code>runProcedure</code> returns immediately or waits until the job completes.</td>
</tr>
<tr>
<td>*** pollInterval is used and timeout is not used, pollInterval will timeout in 60 seconds.***</td>
<td></td>
</tr>
<tr>
<td>pollInterval</td>
<td>If this option is not specified, <code>runProcedure</code> returns immediately. If it is specified, <code>runProcedure</code> waits until the job completes. This argument requires setting a value in seconds to determine how often <code>ectool</code> queries the Commander server for job status, but this is not an indefinite activity - set the <code>timeout</code> value to extend the <code>pollInterval</code> for longer than 60 seconds if needed.</td>
</tr>
<tr>
<td>timeout</td>
<td>This argument requires a value set in <code>seconds</code>. If <code>pollInterval</code> is specified, this timeout causes <code>runProcedure</code> to stop waiting for the job to complete. It does not stop the job itself. If <code>pollInterval</code> is used and <code>timeout</code> is not used, <code>pollInterval</code> will timeout in 60 seconds.</td>
</tr>
</tbody>
</table>

### Positional arguments

- `projectName`, `procedureName`, `scheduleName`

### Response

- The new job ID number.

#### ec-perl

**syntax:** `$cmdr->runProcedure(<project name>, {<optionals}>);`

**Example**

```perl
$cmdr->runProcedure("Sample Project", {procedureName => "Delay", actualParameter => {actualParameterName => "Delay Time", value => 10}}); $xpath = $ec->runProcedure("BSHTest", {procedureName => "FakeMotoBuild", actualParameter => [ {actualParameterName => "builddir", value => $cwd}, {actualParameterName => "board", value => $board}, {actualParameterName => "myview", value => $cwv}, {actualParameterName => "resourcetouse", value => $resourcetouse}, ]});
```

#### ectool

**syntax:** `ectool runProcedure <project name> <procedureName> ...`

**Examples**

```bash
ectool runProcedure <project name> --procedureName <procedure name> --scheduleName <schedule name>
```

```bash
ectool runProcedure "Sample Project" --procedureName "Delay"
```
--actualParameter "Delay Time=10"

**setJobName**

Sets the name of a running job.

You must specify **jobId** and **newName**.

**Notes:**

The **jobId** can be omitted if the command is run as part of an ElectricCommander step.

A job cannot be renamed after it has completed.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>jobId</strong></td>
<td>The ID or name of the job you want to rename. The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td><strong>newName</strong></td>
<td>Supply any name of your choice to rename the job.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**jobId**, **newName**

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* 
```
$cmdr->setJobName(<jobId>, <newName>);
```

*Examples*

```
$cmdr->setJobName(4fa765dd-73f1-11e3-b67e-b0a420524153, "Delay Test_541"); (from the command line)

$cmdr->setJobName("TestJob_252"); (from a step's command)
```

**ectool**

*Syntax:* 
```
ectool setJobName <jobId> <newName> ...
```

*Examples*

```
ectool setJobName 4fa765dd-73f1-11e3-b67e-b0a420524153 "Delay Test"_541 (from the command line)

ectool setJobName "TestJob"_252
```
External Job APIs

What are external job APIs and do you need them?

Overview

ElectricCommander includes a powerful built-in scheduler for both managing execution and real-time reporting for a "running" process. Most Commander Installations choose to use its built-in scheduler because it is more powerful than most in-house built and other scheduling solutions.

However, there are use cases where an external scheduler may be appropriate, for example, an LSF Grid engine. Often, such systems are quite mature and may have been in use for many years. An organizations reliance on an LSF Grid system can mandate it remain as the driving scheduler. Many schedulers lack the richness in their graphical user interface, which is an area where Commander excels—especially as it applies to monitoring the status of complex processes and workflows as they progress in real-time through the Commander system. The Commander GUI also provides powerful auditing capabilities for reviewing results of complex process runs.

External Job APIs are designed to leverage the Commander GUI to display results for jobs running on external schedulers. The external scheduler can issue calls through these APIs to provide a representation of this same job within the Commander Jobs page. Commander users and the external scheduler can then monitor the complete integrated system through a single interface—the Commander GUI.

The external system need not be a formal scheduler. In fact, even a simple script might be able to leverage the External Job Step API. For example, a build script could issue API calls at its beginning and end so the build is represented in Commander as a job.

Using the API does NOT consume agent resources. The API simply allows for graphical representation of external jobs within Commander.

**completeJob**

Completes an externally managed job. Marks the job's root step so the job is marked "completed" when all child steps are completed, and updates the run time for the root step.

You must specify a **jobId**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>jobId</strong></td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td><strong>force</strong></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
outcome | Possible values for outcome:

* success - The job finished successfully.
* warning - The job completed with no errors, but encountered some suspicious conditions.
* error - The job has finished execution with errors.

If specified, the outcome overrides any previously propagated outcome value.

**Positional arguments**

- **jobId**

**Response**

None or status OK message.

**ec-perl**

* syntax: $cmdr->completeJob(<jobId>);

* Example

```perl
$cmdr->completeJob(1234);
```

**ectool**

* syntax: ectool completeJob <jobId>

* Example

```bash
ectool completeJob 1234
```

**completeJobStep**

Completes an externally managed job step. Marks the job step "completed" when all child steps are completed and updates the step run time.

You must specify a **jobStepId**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>exitCode</td>
<td>The step's exit code.</td>
</tr>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>outcome</td>
<td>Possible values for outcome:</td>
</tr>
<tr>
<td></td>
<td>success - The job step finished successfully.</td>
</tr>
<tr>
<td></td>
<td>warning - The job step completed with no errors, but encountered some suspicious conditions.</td>
</tr>
<tr>
<td></td>
<td>error - The job step has finished execution with errors.</td>
</tr>
<tr>
<td></td>
<td>skipped - The job step was skipped.</td>
</tr>
</tbody>
</table>

**Positional arguments**

jobStepId

**Response**

None or status OK message.

**ec-perl**

*syntax:* `$cmdr->completeJobStep(<jobStepId>);`

*Example*

```perl
$cmdr->completeJobStep(5da765dd-73f1-11e3-b67e-b0a420524153);
```

**ectool**

*syntax:* `ectool completeJobStep <jobStepId>`

*Example*

```bash
ectool completeJobStep 5da765dd-73f1-11e3-b67e-b0a420524153
```

**createJob**

Creates an externally managed job that will serve as a container for external job steps.

You must specify `projectName` or `destinationProject`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationProject</td>
<td>If specified, determines the project where the job will reside. You must have modify permission on the destination project. projectName or destinationProject must be specified to determine the project where the job is created, destinationProject is preferred.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| jobNameTemplate            | If specified, the job name will be generated by expanding this argument value.  
                             | **Note:** The job name is generated by expanding the jobNameTemplate argument or the jobNameTemplate from the procedure or the system default. |
| procedureName              | If specified, projectName and procedureName are used as a template for the job. You must have execute permission on the procedure.  
                             | **Note:** The job name is generated by expanding the jobNameTemplate argument or the jobNameTemplate from the specified procedure or the system default. |
| projectName                | The name of the project where this job will reside. You must have modify permission on the destination project.  
                             | projectName or destinationProject must be specified to determine the project where the job is created. If both are specified, destinationProject is preferred. |
| status                     | <pending|runnable|scheduled|running>  
                             | The status argument determines the "starting" job status. This is useful if you want to immediately go into a specific status without having to use modifyJob to update the status. Defaults to pending.  
                             | **Possible values for status:**  
                             | pending - The job is not yet runnable.  
                             | runnable - The job is ready to run.  
                             | scheduled - The job is scheduled to run.  
                             | running - The job is executing. |

### Positional arguments

None

### Response

The new job ID number.

**ec-perl**

**syntax:** $cmdr->createJob({<optionals>});

**Example**

$cmdr->createJob({projectName => "Sample Project"});

**ectool**

**syntax:** ectool createJob ...

---

**API Commands - Job Management**

169
Example

tool createJob --projectName "Sample Project"

createJobStep

Use this API to add Commander managed job steps to a running job or job step as well as to create externally managed steps (if "external" is set).

You must specify the parent job step using either the jobStepId or parentPath arguments (COMMANDER_JOBSTEPID implicitly sets jobStepId). The parent job step status must not be completed.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure. For more information about parameters, click here.</td>
</tr>
<tr>
<td>alwaysRun</td>
<td>If set to 1, indicates this job step will run even if the job is aborted before the job step completes. A useful argument for running a &quot;cleanup&quot; step that should run whether the job step is successful or not. The value for alwaysRun is a &lt;Boolean flag - 0</td>
</tr>
<tr>
<td>broadcast</td>
<td>Use this flag to run the same job step on several resources at the same time. The job step is &quot;broadcast&quot; to all resources listed in the resourceName argument. The broadcast value = &lt;Boolean flag - 0</td>
</tr>
<tr>
<td>condition</td>
<td>If empty or non-zero, the job step will run. If set to &quot;0&quot;, the job step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the job steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps.</td>
</tr>
<tr>
<td>credential</td>
<td>Refers to one or more credentials to attach to this job step. These are &quot;dynamic&quot; credentials, captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded. For more information about credentials, see the Credentials and User Impersonation Help topic.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>credentialName</td>
<td>The credential to use for impersonation on the agent. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>errorHandling</td>
<td>Determines what happens to the procedure if the step fails:</td>
</tr>
<tr>
<td>exclusive</td>
<td>If set to 1, indicates this job step should acquire and retain this resource exclusively. The value for exclusive is a &lt;Boolean flag&gt; -0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>exclusiveMode</td>
<td>Use one of the following options:</td>
</tr>
<tr>
<td></td>
<td>• None - the &quot;default&quot;, which does not retain a resource.</td>
</tr>
<tr>
<td></td>
<td>• Job - keeps the resource for the duration of the job step. No other job can use this resource, regardless of its step limit, until this job completes or &quot;Release Exclusive&quot; is used in a job step. Future steps for this job will use this resource in preference to other resources--if this resource meets the needs of the job steps and its step limit is not exceeded.</td>
</tr>
<tr>
<td></td>
<td>• Step - keeps the resource for the duration of the job step.</td>
</tr>
<tr>
<td></td>
<td>• Call - keeps the resource for the duration of the procedure that called this job step, which is equivalent to 'job' for top level steps.</td>
</tr>
<tr>
<td>external</td>
<td>If set, indicates this job step is an external step. Commander will not schedule or run agent commands for external steps, but instead, represents a step managed outside of Commander.</td>
</tr>
<tr>
<td></td>
<td>The typical usage is with an external Job (see createJob). The status of an external job step is set using modifyJobStep, and it can be completed using completeJobStep. The value for external is a `&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>
**Precondition** property (if it exists) is copied to the job step when the step is created. When the job step is eligible to transition from pending to runnable, the precondition is evaluated. If the precondition result is empty, `false`, or "0", the step remains in the pending state. Any other value allows the step to proceed to the runnable state.

**Note:** A precondition property allows steps to be created with "pause", which then pauses the procedure. In a paused state, all currently running steps continue, but no additional steps will start.

Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a precondition is evaluated. A precondition is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a "0" or "false" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.

Precondition example:
Assume we defined these 4 steps:

1. Build object files and executables
2. Build installer
3. Run unit tests
4. Install bits on test system

Step 1 is an ordinary serial step. Steps 2 and 3 can run in parallel because they depend only on step 1's completion. Step 4 depends on step 2, but not step 3.

You can achieve optimal step execution order with preconditions:
- Make steps 2-4 run in parallel.
- Step 2 needs a job property set at the end of its step to indicate step 2 is completing
  `/myJob/buildInstallerCompleted=1`.
- Set a precondition in step 4:
  `$[/myJob/buildInstallerCompleted]`

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>precondition</strong></td>
<td>The precondition property (if it exists) is copied to the job step when the step is created. When the job step is eligible to transition from pending to runnable, the precondition is evaluated. If the precondition result is empty, <code>false</code>, or &quot;0&quot;, the step remains in the pending state. Any other value allows the step to proceed to the runnable state. <strong>Note:</strong> A precondition property allows steps to be created with &quot;pause&quot;, which then pauses the procedure. In a paused state, all currently running steps continue, but no additional steps will start. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a precondition is evaluated. A precondition is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</td>
</tr>
<tr>
<td><strong>procedureName</strong></td>
<td>The name of the procedure that will contain this job step.</td>
</tr>
<tr>
<td><strong>projectName</strong></td>
<td>The name of the project that contains the procedure where you are adding a new job step.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>releaseExclusive</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>releaseMode</td>
<td>Use one of the following options:</td>
</tr>
<tr>
<td></td>
<td>• None - the &quot;default&quot; - no action if the resource was not previously marked as &quot;retain&quot;.</td>
</tr>
<tr>
<td></td>
<td>• Release - releases the resource at the end of this job step. If the resource for the job step was previously acquired with &quot;Retain exclusive&quot; (either by this job step or some preceding job step), the resource exclusivity is canceled at the end of this job step. The resource is released in the normal way so it may be acquired by other jobs.</td>
</tr>
<tr>
<td></td>
<td>• Release to job - allows a job step to promote a &quot;step exclusive&quot; resource to a Job exclusive resource.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource you want this job step to use.</td>
</tr>
<tr>
<td>shell</td>
<td>Where <code>shell</code> is the name of a program used to execute commands contained in the &quot;command&quot; field. The name of a temporary file containing commands will be appended to the end of this invocation line. Normally, this file is a command shell, but it can be any other command line program. The default is to use the standard shell for the platform it runs on (cmd for Windows, sh for UNIX). Applicable to command steps only.</td>
</tr>
<tr>
<td>status</td>
<td>`&lt;pending</td>
</tr>
<tr>
<td></td>
<td>pending - The job step is not yet runnable.</td>
</tr>
<tr>
<td></td>
<td>runnable - The job step is ready to run.</td>
</tr>
<tr>
<td></td>
<td>scheduled - The job step is scheduled to run.</td>
</tr>
<tr>
<td></td>
<td>running - The job step is executing.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the new job step you are creating. You can use any name of your choice.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>The name of the nested procedure to call when this job step runs. If a subprocedure is specified, do not include the <code>command</code> or <code>commandFile</code> options.</td>
</tr>
</tbody>
</table>
### API Commands - Job Management

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>subproject</td>
<td><strong>If a subprocedure argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.</strong></td>
</tr>
<tr>
<td>timeLimit</td>
<td>The maximum length of time the job step is allowed to run. After the time specified, the job step will be aborted. The time limit is specified in units that can be hours, minutes, or seconds.</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>Specify hours</td>
</tr>
<tr>
<td>workingDirectory</td>
<td>The Commander agent sets this directory as the “current working directory,” when running the command contained in the job step. If no working directory is specified in the job step, Commander uses the directory it created for the job in the Commander workspace as the working directory. <strong>Note:</strong> If running a job step on a proxy resource, this directory must exist on the proxy target.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace where this job step's log files will be stored.</td>
</tr>
</tbody>
</table>

#### Positional arguments

- jobStepId or parentPath

#### Response

Returns a jobStep object.

#### ec-perl

**syntax:** $cmdr->createJobStep({<optionals>});

**Examples**

```perl
$cmdr->createJobStep ({parentPath => "/jobs/123", external => "1"});

$cmdr->createJobStep ({jobStepId => "5da765dd-73f1-11e3-b67e-b0a420524153", external => "1"});

# Create a job step that calls a subprocedure and passes a parameter credential
# 'coolProcedure' is a procedure within the Default project with one parameter
# credential named 'sshCredentialParameter'.
$cmdr->createJobStep(
  {
    projectName => 'Default',
    subprocedure => 'coolProcedure',
    actualParameter => [
      {
        
```
actualParameterName => 'sshCredentialParameter',
value => 'sshCredentialParameter'
]
],

credential => [
{
  credentialName => 'sshCredentialParameter',
  userName => 'sshUser',
  password => 'super_secure_sshPassword'
}
]
];

# Create two parallel job steps and send them to the Commander server using the batch API.

# Create the batch API object
my $batch = $ec->newBatch('parallel');

# Create multiple requests
my @reqIds = {
  $batch->createJobStep(
    {
      parallel => '1',
      projectName => 'Default',
      subprocedure => 'coolProcedure',
      actualParameter => [
        {
          actualParameterName => 'input',
          value => 'helloWorld'
        }
      ],
    }
  ),
  $batch->createJobStep(
    {
      parallel => '1',
      projectName => 'Default',
      subprocedure => 'coolProcedure',
      actualParameter => [
        {
          actualParameterName => 'input',
          value => 'helloWorld'
        }
      ],
    }
  ),
};
```json
{
    parallel => '1',
    projectName => 'Default',
    subprocedure => 'coolProcedure',
    actualParameter => [
        {
            actualParameterName => 'input',
            value => 'helloWorld'
        }
    ],
}
```

# Send off the requests

$batch->submit();

**ectool**

*Syntax:* ectool createJobStep ...

**Examples**

ectool createJobStep --parentPath /jobs/123 --external 1

ectool createJobStep --jobStepId 5da765dd-73f1-11e3-b67e-b0a420524153 --external 1

ectool createJobStep --parallel 1 --projectName Default --subprocedure coolProcedure --actualParameter input=helloWorld

**modifyJob**

Modifies the status of an externally managed job.

You must specify a **jobId**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
</tbody>
</table>
### modifyJob

Modifies the status of an externally managed job.

You must specify a `projectName` and `jobStepId`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobStepId</td>
<td>The Commander-generated ID for the job step.</td>
</tr>
</tbody>
</table>

### modifyJobStep

Modifies the status of an externally managed job step.

You must specify a `projectName` and `jobStepId`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobStepId</td>
<td>The Commander-generated ID for the job step.</td>
</tr>
</tbody>
</table>
### Arguments and Descriptions

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| status | `<pending|runnable|scheduled|running>`

The `status` argument determines the current status of the job, and also sets related timing values.

Possible values for `status`:
- `pending` - The job step is not yet runnable.
- `runnable` - The job step is ready to run.
- `scheduled` - The job step is scheduled to run.
- `running` - The job step is executing.

### Positional arguments

- **jobStepId**

### Response

Returns a modified `jobStep` object.

**ec-perl**

`syntax:` 

```perl
$cmdr->modifyJobStep (<jobStepId>, {<optional>});
```

**Example**

```perl
$cmdr->modifyJobStep (4fa765dd-73f1-11e3-b67e-b0a420524153, {status => "running"});
```

**ectool**

`syntax:` 

```bash
ectool modifyJobStep <jobStepId> ...
```

**Example**

```bash
ectool modifyJobStep 4fa765dd-73f1-11e3-b67e-b0a420524153 --status "running"
```

### waitForJob

Waits until the specified job reaches a given status or the timeout expires. Returns the result from the final `getJobStatus` query.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The job to wait for. The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>finalStatus</td>
<td>The status to wait for. Must be either &quot;running&quot; or &quot;completed&quot; (default is &quot;completed&quot;).</td>
</tr>
<tr>
<td>timeout</td>
<td>The number of seconds to wait before giving up on a request.</td>
</tr>
</tbody>
</table>
**Positional arguments**

jobId

**Response**

Returns the result from the final `getJobStatus` query.

**ec-perl**

**syntax:**

```perl
$cmdr->waitForJob ($4fa765dd-73f1-11e3-b67e-b0a420524153, {<optional>});
```

**Example**

```perl
$cmdr->waitForJob (4fa765dd-73f1-11e3-b67e-b0a420524153, {status => "running"});
```

*Back to Top*
API Commands - Parameter Management

**attachParameter**

Attaches a formal parameter to a step.

Attaching a parameter allows a step to use the credential (passed in a parameter) as an actual parameter to a subprocedure call or directly in a getFullCredential call in a command step. For more information about parameters, click here.

You must specify **projectName, procedureName, stepName, and formalParameterName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>formalParameterName</td>
<td>The name of the procedure’s parameter, containing a credential reference.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the step.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step to attach the parameter credential.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**projectName, procedureName, stepName, formalParameterName**

**Response**

None or status OK message.

**ec-perl**

**syntax:**

```
$cmdr->attachParameter(<projectName>, <procedureName>, <stepName>, <formalParameterName>);
```
Example

$cmdr->attachCredential("Test Proj", "Run Build", "Get Sources", "SCM Credential");

dcreateAtualParameter

createActualParameter

Creates a new actual parameter for a step that calls a nested procedure. The parameter is passed to the nested procedure when the step runs. At run time, the actual parameter name must match the name of a formal parameter in the nested procedure.

Passing Actual Parameters

You can use actual parameters in three types of API calls:

- calling runProcedure to start a new job
- setting up a schedule
- creating or modifying a subprocedure step

For example, when you call runProcedure using ectool, set the actual parameters to the procedure on the command line using the optional argument --actualParameter, followed by a list of name/value pairs.

The following is an example of calling a procedure named MasterBuild:

```bash
ectool runProcedure "project A" --procedureName "MasterBuild"
--actualParameter Branch=main Type=Debug
```

To make this call using the Perl API, define a list. Each element of the list is an anonymous hash reference that specifies one of the actual parameters. Now you can pass a reference to the list as the value of the actualParameter argument.

Here is the same example called via the Perl API:

```perl
# Run the procedure
$urlPath = $cmdr->runProcedure("project A",
    {procedureName => "MasterBuild",
      actualParameter => [
        {actualParameterName => 'Branch',
          value => 'main'},
        {actualParameterName => 'Type',
          value => 'Debug'}],
    });
```

Specifying most arguments to the createStep API in Perl is fairly intuitive; like any other API, you specify key-value pairs in a hash argument for all optional parameters. However, specifying actual parameters is more involved because actual parameters are not arbitrary key-values characterizing the step. Instead, they are key-values characterizing actual parameters to the step. See the following createStep request in XML:
<createStep>
  <projectName>MyProject</projectName>
  <procedureName>MyProcedure</procedureName>
  <stepName>Step1</stepName>
  <actualParameter>
    <actualParameterName>parm1</actualParameterName>
    <value>myval</value>
  </actualParameter>
  <actualParameter>
    <actualParameterName>parm2</actualParameterName>
    <value>val2</value>
  </actualParameter>
</createStep>

Each actual parameter key-value is under an <actualParameter> element. Code this in the optional arguments hash in the Perl API like this:

```perl
{ ..., => ..., actualParameter => [{actualParameterName => 'parm1',
                                value => 'myval'),
                               {actualParameterName => 'parm2',
                                value => 'val2'}],
        ... => ...}
```

In other words, the value of the actualParameter key in the optional arguments hash is a list of hashes, each representing one actual parameter. If the sub-procedure call takes only one actual parameter, the value of the actualParameter key can be specified as just the hash representing the one parameter:

```perl
actualParameter => {actualParameterName => 'parm1',
                    value => 'myval'}
```

You must specify projectName, procedureName, stepName, and actualParameterName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameterName</td>
<td>The name of the parameter. This name must be unique within the step, and at run time it must match the name of a formal parameter in the subprocedure.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the procedure.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing this parameter.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step that calls a subprocedure.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>value</td>
<td>This value is passed to the subprocedure as the value of the matching formal parameter.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>
**Positional arguments**

projectName, procedureName, stepName, actualParameterName

**Response**

None or status OK message.

**ec-perl**

*Syntax:* $cmdr->createActualParameter(<projectName>, <procedureName>, <stepName>, <actualParameterName>, {<optionals>});

*Example*

$cmdr->createActualParameter("Sample Project", "CallSub", "Step1", "Extra Parm", {value => "abcd efg"});

**ectool**

*Syntax:* ectool <projectName> <procedureName> <stepName> <actualParameterName>

*Example*

ectool createActualParameter "Sample Project" "CallSub" "Step1" "Extra Parm" --value "abcd efg"

**createTimeformalParameter**

Creates a new formal parameter.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt;...&lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>defaultValue</td>
<td>This value is used for the formal parameter if a value is not supplied by the caller.</td>
</tr>
<tr>
<td>expansionDeferred</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>formalParameterName</td>
<td>The name for this parameter - used when the procedure is invoked to specify a value for the parameter.</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
procedureName | The name of the procedure containing the parameter. **Note:** In releases earlier than ElectricCommander 5.0, `procedureName` is required. In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, `procedureName` is optional.

projectName | The name of the project containing the procedure.

required | `<Boolean flag - 0|1|true|false>` If set to 1, this value indicates whether a non-blank value must be supplied when calling the procedure.

stateDefinitionName | The name of the state definition.

type | The "type" can be any string value. Used primarily by the web interface to represent custom form elements. However, if "credential" is the string value, the server will expect a credential as the parameter value.

workflowDefinitionName | The name of the workflow definition.

**Positional arguments**

In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, for procedure parameters: `projectName` and `formalParameterName`.

In releases earlier than ElectricCommander 5.0, for procedure parameters: `projectName`, `procedureName`, and `formalParameterName`.

For workflow state parameters: `projectName`, `formalParameterName`, `workflowDefinitionName` and `stateDefinitionName`

**Response**

None or status OK message.

**ec-perl**

**syntax:** $cmdr->createFormalParameter(<projectName>, <formalParameterName>, {<optionals>});

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

**syntax:** $cmdr->createFormalParameter(<projectName>, <procedureName>, <formalParameterName>, {<optionals>});

**Example**

$cmdr->createFormalParameter("Sample Project", "Branch Name", {required => 1 });

**Examples using parameters to create checkbox, radio button, and dropdown box**

Checkbox example:
$ec->createFormalParameter(
    $newProjectName,
    "$buildprocedurename",
    'CheckoutSources',
    [
        type => 'checkbox',
        required => 0,
        defaultValue => 'true',
        description => "If checked, update the sandbox from Subversion (turn off for debugging only)."
    ]);
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ec_customEditorData/parameters/CheckoutSources/checkedValue", "true");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ec_customEditorData/parameters/CheckoutSources/uncheckedValue", "false");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ec_customEditorData/parameters/CheckoutSources/initiallyChecked", "0");

**Radio button example:**

$ec->createFormalParameter(
    $newProjectName,
    "$buildprocedurename",
    'BuildType',
    [
        type => "radio",
        required => 1,
        defaultValue => 'Continuous',
        description => "Select type of build"
    ]);
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ec_customEditorData/parameters/BuildType/options/optionCount", "2");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ec_customEditorData/parameters/BuildType/options/type", "list");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ec_customEditorData/parameters/BuildType/options/option1/text", "one");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ec_customEditorData/parameters/BuildType/options/option1/value", "1");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ec_customEditorData/parameters/BuildType/options/option2/text", "two");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ec_customEditorData/parameters/BuildType/options/option2/value", "2");

**Dropdown menu example:**

$ec->createFormalParameter(
    $newProjectName,
    "$buildprocedurename",
    'BuildType',
    [
        type => "dropdown",
        required => 1,
        defaultValue => 'Continuous',
        description => "Select type of build"
    ]);

$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ 
  ec_customEditorData/parameters/BuildType/options/optionCount", "2");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ 
  ec_customEditorData/parameters/BuildType,options/option1/text", "one");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ 
  ec_customEditorData/parameters/BuildType/options/option2/text", "two");
$ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/ 
  ec_customEditorData/parameters/BuildType/options/option2/value", "2");

**ectool**

For procedure parameters

**syntax:** ectool createFormalParameter <projectName> <formalParameterName> ...

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

**syntax:** ectool createFormalParameter <projectName> <procedureName> <formalParameterName> ...

**Example**

ectool createFormalParameter "Sample Project" "Branch Name" --required 1

For workflow state parameters

**syntax:** ectool createFormalParameter --formalParameterName <name> --projectName <name> --workflowDefinitionName <name> --stateDefinitionName <name>

**deleteActualParameter**

Deletes an actual parameter.

You must specify a **projectName, procedureName, stepName, and actualParameterName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameterName</td>
<td>The name of the actual parameter you want to delete.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that contains the step with this parameter.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains this actual parameter.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing the actual parameter.</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
stateDefinitionName | The name of the state definition.
stepName | The name of the step that contains this actual parameter you want to delete.
transitionDefinitionName | The name of the transition definition.
workflowDefinitionName | The name of the workflow definition.

**Positional arguments**
- `projectName`, `procedureName`, `stepName`, `actualParameterName`

**Response**
None or a status OK message.

**ec-perl**
*Syntax:* `$cmdr->deleteActualParameter(<projectName>, <procedureName>, <stepName>, <actualParameterName>);`

*Example*
```
$cmdr->deleteActualParameter('Sample Project', 'CallSub', 'Step1', 'Different Param');
```

**ectool**
*Syntax:* `ectool deleteActualParameter <projectName> <procedureName> <stepName> <actualParameterName>`

*Example*
```
ectool deleteActualParameter "Sample Project" "CallSub" "Step1" "Different Parm"
```

**deleteFormalParameter**
Deletes a formal parameter.
You must specify `projectName` and `formalParameterName`.

Arguments | Descriptions
--- | ---
formalParameterName | The name of the formal parameter you want to delete.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>The name of the procedure that contains this parameter. <strong>Note:</strong> In releases earlier than ElectricCommander 5.0, <code>procedureName</code> is required. In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, <code>procedureName</code> is optional.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains the procedure/parameter you want to delete.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, `projectName` and `formalParameterName`.

In releases earlier than ElectricCommander 5.0, `projectName`, `procedureName`, and `formalParameterName`.

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* $cmdr->deleteFormalParameter(<projectName>, <formalParameterName>);

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

*Syntax:* $cmdr->deleteFormalParameter(<projectName>, <procedureName>, <formalParameterName>);

**Example**

$cmdr->deleteFormalParameter("Sample Project", "Build Name");

**ectool**

*Syntax:* ectool deleteFormalParameter <projectName> <formalParameterName>

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

*Syntax:* ectool deleteFormalParameter <projectName> <procedureName> <formalParameterName>

**Example**

ectool deleteFormalParameter "Sample Project" "Build Name"
**detachParameter**

Detaches a formal parameter from a step.

You must specify **projectName**, **procedureName**, **stepName**, and **formalParameterName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>formalParameterName</td>
<td>The name of the parameter to detach.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that contains this parameter.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains this parameter.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step where this parameter is currently attached.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**projectName**, **procedureName**, **stepName**, **formalParameterName**

**Response**

None or a status OK message.

**ec-perl**

**syntax**: `ecmdr->detachParameter(<projectName>, <procedureName>, <stepName>, <formalParameterName>);

**Example**

```perl
ecmdr-> detachParameter("Test Proj", "Run Build", "Get Sources", "SCM Credential");
```

**ectool**

**syntax**: `ectool detachParameter <projectName> <procedureName> <stepName> <formalParameterName>`

**Example**

```tools
ectool detachParameter "Test Proj" "Run Build" "Get Sources" "SCM Credential"
```

**getActualParameter**

Retrieves an actual parameter by its name. For more information about parameters, click [here](#).

You must specify an **actualParameterName**. If you need actual parameters on a step, the following 3 arguments must be used together to specify a step: **projectName**, **procedureName**, and **stepName**.
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameterName</td>
<td>The name of the actual parameter.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application, if the actual parameter is on an application process step; must be unique among all projects.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component, if the actual parameter is on a component process step.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created. Supply this argument to query a subprocedure call to the job step's parameter.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure to query for the procedure step's parameter.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the actual parameter is on a process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the actual parameter is on a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project to query for a schedule or procedure step's parameter.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule to query for the schedule's parameter.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step to query for the step's parameter.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

## Positional arguments

- actualParameterName

## Response

- One `actualParameter` element.
**getActualParameters**

Retrieves all actual parameters from a job, job step, schedule, or step. For more information about parameters, click here.

You must specify object locators to find the parameter. If finding parameters on a step, you must use **projectName**, **procedureName**, and **stepName** to specify a step.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application, if the actual parameter is on an application process step; must be unique among all projects.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component, if the actual parameter is on a component process step.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing these parameters.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing these parameters.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the actual parameter is on a process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the actual parameter is on a process step.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing parameters.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step containing parameters.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**
Arguments to locate the parameter, beginning with the top-level object locator.

**Response**
Zero or more `actualParameter` elements.

**ec-perl**

*Syntax:* `$cmdr->getActualParameters({...});`

*Example*

```perl
$cmdr->getActualParameters({"projectName" => "Sample Project",
                           "procedureName" => "CallSub",
                           "stepName" => "Step1"});
```

**ectool**

*Syntax:* `ectool getActualParameters ...`

*Example*

```bash
ectool getActualParameters --projectName "Sample Project"
                           --procedureName "CallSub" --stepName "Step1"
```

Back to Top

**getFormalParameter**

Retrieves a formal parameter by its name.

You must specify `projectName` and `formalParameterName`. 
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>formalParameterName</td>
<td>The name of the formal parameter.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the formal parameter. Note: In releases earlier than ElectricCommander 5.0, procedureName is required. In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, procedureName is optional.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the procedure.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**

In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, `projectName` and `formalParameterName`.

In releases earlier than ElectricCommander 5.0, `projectName`, `procedureName`, and `formalParameterName`.

**Response**

One `formalParameter` element.

**ec-perl**

*Syntax*: `cmdr->getFormalParameter('<projectName>', '<formalParameterName>');`

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

*Syntax*: `cmdr->getFormalParameter('<projectName>', '<procedureName>', '<formalParameterName>');`

**Example**

```
$cmdr->getFormalParameter("Test", "Get Sources");
```

**ectool**

*Syntax*: `ectool getFormalParameter<projectName> <formalParameterName>`

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

*Syntax*: `ectool getFormalParameter<projectName> <procedureName> <formalParameterName>`

**Example**

```
ectool getFormalParameter Test "Get Sources"
```

Back to Top
getFormalParameters

Retrieves all formal parameters from a procedure, schedule, or step.

You must specify locator arguments to identify a procedure, schedule, or subprocedure step. If the locators identify a schedule or step, the formal parameters of the called procedure are returned.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>The name of the procedure. <em>Also requires</em> the projectName</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the object whose parameters are being retrieved. <em>Also requires</em> the projectName</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule. <em>Also requires</em> the projectName</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step. <em>Also requires</em> the projectName and procedureName</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**

Arguments to locate the formal parameter, beginning with the top-level object locator.

**Response**

An XML stream containing zero or more formalParameter elements.

**ec-perl**

*syntax:* $cmdr->getFormalParameters(<projectName>, {<optionals>});

*Example*

$cmdr->getFormalParameters("Test", {procedureName => "Build"});

**ectool**

*syntax:* ectool getFormalParameters <projectName> ...

*Example*

getFormalParameters Test --procedureName Build

Back to Top
**modifyActualParameter**

Modifies an existing actual parameter. An actual parameter is a name/value pair passed to a subprocedure. This command supports renaming the actual parameter and setting its value. For more information about parameters, click [here](#).

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameterName</td>
<td>The name of the actual parameter to modify.</td>
</tr>
<tr>
<td>newName</td>
<td>Supply a name of your choice to rename the parameter.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the step with this parameter.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing this parameter.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step containing this parameter.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>value</td>
<td>Changes the current value on an actual parameter. This value is passed to the subprocedure as the value of the matching formal parameter.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `$cmdr->modifyActualParameter(<projectName>, <procedureName>, <stepName>, <actualParameterName>, {<optionals>});`

**Example**

```perl
$cmdr->modifyActualParameter("Sample Project", "CallSub", "Step1", "Extra Parm", 
{newName => "myParm"});
```

**ectool**

*Syntax:* `ectool modifyActualParameter <projectName> <procedureName> <stepName> 
<actualParameterName> ...`
modifyActualParameter

Example
ectool modifyActualParameter "Sample Project" "CallSub" "Step1" "Extra Parm"
   --newName "Different Parm"

modifyFormalParameter

Modifies an existing formal parameter.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultValue</td>
<td>This value is used for the formal parameter if one is not supplied by the caller.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>expansionDeferred</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>formalParameterName</td>
<td>The name for this formal parameter. Used when the procedure is invoked to specify a value for the parameter.</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the parameter.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing this parameter.</td>
</tr>
<tr>
<td></td>
<td>Note: In releases earlier than ElectricCommander 5.0, procedureName is required. In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, procedureName is optional.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing this parameter.</td>
</tr>
<tr>
<td>required</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>type</td>
<td>type can be any string value. Used primarily by the web interface to represent custom form elements. However, if &quot;credential&quot; is the string value, the server will expect a credential as the parameter value.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>
**Positional arguments**

In ElectricCommander 5.0 and later and in ElectricFlow 5.0 and later, for procedure parameters:

- **projectName** and **formalParameterName**.

In releases earlier than ElectricCommander 5.0, for **procedure parameters**: **projectName**, **procedureName**, and **formalParameterName**.

For **workflow state parameters**: **projectName**, **formalParameterName**, **workflowDefinitionName** and **stateDefinitionName**

**Response**

None or a status OK message.

**ec-perl**

```perl
syntax: $cmdr->modifyFormalParameter(<projectName>, <formalParameterName>, {<optionals>});
```

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

```perl
syntax: $cmdr->modifyFormalParameter(<projectName>, <procedureName>, <formalParameterName>, {<optionals>});
```

**Example**

```
$cmdr->modifyFormalParameter("Sample Project", "Branch Name", 
    {defaultValue => "main"});
```

**ectool**

For **procedure parameters**:

```bash
tenet modifyFormalParameter <projectName> <formalParameterName> ...
```

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

```bash
tenet modifyFormalParameter <projectName> <procedureName> <formalParameterName> ...
```

**Example**

```
tenet modifyFormalParameter "Sample Project" "Branch Name" 
    --defaultValue main
```

For **workflow state parameters**:

```bash
tenet modifyFormalParameter --formalParameterName <name>  
    --projectName <name>  --workflowDefinitionName <name>  --stateDefinitionName <name>
```

Back to Top
API Commands - Plugin Management

- deletePlugin
- getPlugin
- getPlugins
- installPlugin
- modifyPlugin
- promotePlugin
- uninstallPlugin

**deletePlugin**

Deletes an existing plugin object without deleting the associated project or files.

You must specify a `pluginName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginName</td>
<td>The name of the plugin you want to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`pluginName`

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `$cmdr->deletePlugin(<pluginName>);`

*Example*:

```
$cmdr->deletePlugin("TheWidget-1.0");
```

**ectool**

*Syntax:* `ectool deletePlugin <pluginName>`

*Example*:

```
ectool deletePlugin TheWidget-1.0
```

**getPlugin**

Retrieves an installed plugin.

You must specify the `pluginName`.
Arguments | Descriptions
---|---
pluginName | The name of the plugin to find. If the name is specified without a version number, the currently promoted version is returned if possible.

**Positional arguments**

pluginName

**Response**

One plugin element, which includes the plugin ID, name, time created, label, owner, key, version, and more.

**ec-perl**

*syntax:* `$cmdr->getPlugin(<pluginName>);

*Example*

```
$cmdr->getPlugin("TheWidget");
```

**ectool**

*syntax:* `ectool getPlugin <pluginName>

*Example*

```
ectool getPlugin TheWidget
```

**getPlugins**

Retrieves all installed plugins.

| Arguments | Descriptions |
---|---|
None | |

**Positional arguments**

None

**Response**

Zero or more plugin elements.

**ec-perl**

*syntax:* `$cmdr->getPlugins();

*Example*

```
$cmdr->getPlugins();
```
###.installPlugin

Installs a plugin from a JAR file. Extracts the JAR contents on the server and creates a project and a plugin.

You must specify the `url`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>url</td>
<td>The location of the plugin JAR file to install. If the location refers to a file on the client machine, the file will be uploaded to the server. If the location refers to a remote accessible file (for example, via an <code>http://url</code>), the server will download it. If the location is a <code>file:</code> reference, the file will be read directly from the specified location on the server's file system.</td>
</tr>
</tbody>
</table>

#### Positional arguments

`url`

#### Response

One `plugin` element.

### ec-perl

```
syntax: $cmdr->installPlugin(<url>, {...});

Example

$cmdr->installPlugin("./myPlugin.jar")
```

### ectool

```
syntax: ectool installPlugin <url> ...

Example

ectool installPlugin ./myPlugin.jar
```

Back to Top
modifyPlugin

Modifies an existing plugin.

**Note:** Some plugin attributes available on the Plugins web page are not available in any of the plugin-related APIs. Because some plugin meta data comes from the plugin.xml file, the web server can access this data, but the Commander server cannot. Thus, the Plugin Manager, run in the web server context, provides additional information and functionality.

You must specify the **pluginName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>The author of the plugin.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must</td>
</tr>
<tr>
<td></td>
<td>surround your text with <code>&lt;html&gt; ...&lt;/html&gt;</code> tags. The only HTML tags allowed</td>
</tr>
<tr>
<td></td>
<td>in the text are: <code>&lt;a&gt;</code> <code>&lt;b&gt;</code> <code>&lt;br&gt;</code> <code>&lt;div&gt;</code> <code>&lt;dl&gt;</code> <code>&lt;i&gt;</code> <code>&lt;li&gt;</code> <code>&lt;ol&gt;</code> <code>&lt;p&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;pre&gt;</code> <code>&lt;span&gt;</code> <code>&lt;style&gt;</code> <code>&lt;table&gt;</code> <code>&lt;tc&gt;</code> <code>&lt;td&gt;</code> <code>&lt;th&gt;</code> <code>&lt;tr&gt;</code> <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>label</td>
<td>The name of the plugin as displayed on the Plugins web page.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin to modify. If the name is specified without a</td>
</tr>
<tr>
<td></td>
<td>version number, the currently promoted version is used if possible.</td>
</tr>
</tbody>
</table>

**Positional arguments**

pluginName

**Response**

One **plugin** element.

**ec-perl**

**syntax:** `$cmdr->modifyPlugin<$pluginName>, {...};`

**Example**

```
$cmdr->modifyPlugin('TheWidget', {description => "new description"});
```

**ectool**

**syntax:** `ectool modifyPlugin <pluginName> ...

**Example**

```
ectool modifyPlugin TheWidget --description "new description"
```

Back to Top
**promotePlugin**

Sets the promoted flag on a plugin. Only one version of a plugin can be promoted at a time, so setting the promoted flag to "true" on one version sets the flag to false on all other plugins with the same key. The promoted version is the one resolved by an indirect reference of the form $[/plugins/<key>] or a plugin name argument without a specified version.

You must specify the **pluginName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PluginName</td>
<td>The name of the plugin to promote. If the name is specified without a version number, the currently promoted version is used if possible.</td>
</tr>
<tr>
<td>Promoted</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

**pluginName**

**Response**

One plugin element, which includes the plugin ID, name, time created, label, owner, key, version, project name, and more.

**ec-perl**

`syntax:`

```perl
$cmdr->promotePlugin(<pluginName>, {<optionals>>};
```

**Example**

```
$cmdr->promotePlugin("TheWidget-1.0");
```

**ectool**

`syntax:`

```bash
  ectool promotePlugin <pluginName> ...
```

**Example**

```
  ectool promotePlugin TheWidget-1.0
```

Back to Top

**uninstallPlugin**

UnInstalls a plugin, deleting the associated project and any installed files.

You must specify the **pluginName**.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginName</td>
<td>The name of the plugin to uninstall. If the name is specified without a version number, the currently promoted version is used if possible.</td>
</tr>
<tr>
<td>timeout</td>
<td>The maximum amount of time to spend waiting for this operation to complete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

pluginName

**Response**

None or a status OK message.

**ec-perl**

* syntax:*

```
$cmdr->uninstallPlugin(<pluginName>, {<optionals>});
```

* Example*

```
$cmdr->uninstallPlugin("TheWidget-1.0");
```

**ectool**

* syntax:*

```
ectool uninstallPlugin <pluginName> ...
```

* Example*

```
ectool uninstallPlugin TheWidget-1.0
```

[Back to Top]
API Commands - Procedure Management

createProcedure
createStep
deleteProcedure
deleteStep
getProcedure
getProcedures
getStep
getSteps
modifyProcedure
modifyStep
moveStep

createProcedure

Creates a new procedure for an existing project.
You must specify projectName and procedureName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>jobNameTemplate</td>
<td>Template used to determine the default name of jobs launched from a procedure.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name you define for this procedure. You can use any name of your choice.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains this procedure.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of a resource or pool to use as the default for steps run by this procedure.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeLimit</td>
<td>If no time limit was specified on the calling step, time limits are copied to the calling step from the procedure. If the procedure is called from runProcedure (or a schedule), the time limit acts as a global job timeout. The &quot;timer&quot; for the procedure starts as soon as the calling step/job becomes runnable (all preconditions are satisfied).</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>Time limit units are hours</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace to use as the default for steps run by this procedure.</td>
</tr>
</tbody>
</table>

#### Positional arguments

projectName, procedureName

#### Response

None or status OK message.

#### ec-perl

**syntax:** $cmdr->createProcedure(<projectName>, <procedureName>, {<optionals>});

**Example**

$cmdr->createProcedure("Test Proj", "Run Build", {resourceName => "Test Resource"});

#### ectool

**syntax:** ectool createProcedure <projectName> <procedureName> ...

**Example**

ectool createProcedure "Test Proj" "Run Build" --resourceName "Test Resource"

Back to Top

### createStep

Use this command to create a new procedure step.

Fundamentally, ElectricCommander supports three types of steps:

- Command Step - the step executes a command or script under the control of a shell program.
- Subprocedure Step - the step invokes another Commander procedure. In this case, the step will not complete until all subprocedure steps have completed.
- Custom Step

You must specify a **projectName**, **procedureName**, and **stepName**.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure. For more information about parameters, click here.</td>
</tr>
<tr>
<td>alwaysRun</td>
<td>If set to 1, indicates this step will run even if the job is aborted before the step completes. A useful argument for running a &quot;cleanup&quot; step that should run whether the job is successful or not. The value for alwaysRun is a &lt;Boolean flag -0</td>
</tr>
<tr>
<td>broadcast</td>
<td>Use this flag to run the same step on several resources at the same time. The step is &quot;broadcast&quot; to all resources listed in the resourceName argument. The broadcast value = &lt;Boolean flag -0</td>
</tr>
<tr>
<td>command</td>
<td>The command to run. This argument is applicable to command steps only.</td>
</tr>
<tr>
<td>commandFile</td>
<td>This option is supported only in Perl and ectool bindings - it is not a part of the XML protocol. Contents of the command file is read and stored in the &quot;command&quot; field. This is an alternative argument for command and is useful if the &quot;command&quot; field spans multiple lines. The commandFile value is the actual command file text. This argument is applicable to command steps only.</td>
</tr>
<tr>
<td>condition</td>
<td>If empty or non-zero, the step will run. If set to &quot;0&quot;, the step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The credential to use for impersonation on the agent. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;projects(BuildProject)/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;ii&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>errorHandling</td>
<td>Determines what happens to the procedure if the step fails:</td>
</tr>
<tr>
<td>exclusive</td>
<td>If set to 1, indicates this step should acquire and retain this resource exclusively. The value for <code>exclusive</code> is a `&lt;Boolean flag -0</td>
</tr>
<tr>
<td>exclusiveMode</td>
<td>Use one of the following options:</td>
</tr>
<tr>
<td>logFileName</td>
<td>A custom log file name produced by running the step. By default, ElectricCommander assigns a unique name for this file.</td>
</tr>
<tr>
<td>parallel</td>
<td>If set, indicates this step should run at the same time as adjacent steps marked to run as parallel also. The value for <code>parallel</code> is a `&lt;Boolean flag -0</td>
</tr>
</tbody>
</table>
## Arguments and Descriptions

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>postProcessor</td>
<td>The name of a program to run after a step completes. This program looks at the step output to find errors and warnings. Commander includes a customizable program called &quot;postp&quot; for this purpose. The value for postProcessor is a command string for invoking a post-processor program in the platform shell for the resource (cmd for Windows, sh for UNIX).</td>
</tr>
<tr>
<td>precondition</td>
<td>By default, if the step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a precondition is evaluated. A precondition is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that will contain this step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains the procedure where you are adding a new step.</td>
</tr>
<tr>
<td>releaseExclusive</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>releaseMode</td>
<td>Use one of the following options:</td>
</tr>
<tr>
<td></td>
<td>• None - the &quot;default&quot; - no action if the resource was not previously marked as &quot;retain&quot;.</td>
</tr>
<tr>
<td></td>
<td>• Release - releases the resource at the end of this step. If the resource for the step was previously acquired with &quot;Retain exclusive&quot; (either by this step or some preceding step), the resource exclusivity is canceled at the end of this step. The resource is released in the normal way so it may be acquired by other jobs.</td>
</tr>
<tr>
<td></td>
<td>• Release to job - allows a step to promote a &quot;step exclusive&quot; resource to a Job exclusive resource.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource you want this step to use.</td>
</tr>
<tr>
<td>shell</td>
<td>Where shell is the name of a program used to execute commands contained in the &quot;command&quot; field. The name of a temporary file containing commands will be appended to the end of this invocation line. Normally, this file is a command shell, but it can be any other command line program. The default is to use the standard shell for the platform it runs on (cmd for Windows, sh for UNIX). This is applicable to command steps only.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the new step you are creating. You can use any name of your choice.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the command or commandFile options.</td>
</tr>
<tr>
<td>subproject</td>
<td>If a subprocedure argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.</td>
</tr>
<tr>
<td>timeLimit</td>
<td>The maximum length of time the step is allowed to run. After the time specified, the step will be aborted. The time limit is specified in units that can be hours, minutes, or seconds.</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>Specify hours</td>
</tr>
<tr>
<td>workingDirectory</td>
<td>The Commander agent sets this directory as the &quot;current working directory,&quot; when running the command contained in the step. If no working directory is specified in the step, Commander uses the directory it created for the job in the Commander workspace as the working directory. Note: If running a step on a proxy resource, this directory must exist on the proxy target.</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
workspaceName | The name of the workspace where this step's log files will be stored.

**Positional arguments**
- projectName, procedureName, stepName

**Response**
None or a status OK message.

**ec-perl**

**syntax:** `cmdr->createStep(<projectName>, <procedureName>, <stepName>, {<optionals>});`

Specifying most arguments to the Perl `createStep` API is fairly intuitive. Similar to any other API, key-value pairs are specified in a hash argument for all optional parameters. However, specifying actual parameters is a little different because they are not arbitrary key-values characterizing the step. Actual parameters are key-values characterizing actual parameters to the step. See the following `createStep` request in XML:

```xml
<createStep>
  <projectName>MyProject</projectName>
  <procedureName>MyProcedure</procedureName>
  <stepName>Step1</stepName>
  <actualParameter>
    <actualParameterName>parm1</actualParameterName>
    <value>myval</value>
  </actualParameter>
  <actualParameter>
    <actualParameterName>parm2</actualParameterName>
    <value>val2</value>
  </actualParameter>
</createStep>
```

Each actual parameter key-value is under an `<actualParameter>` element, which is codified in the optional arguments hash in the Perl API like this:

```perl
{... => ...,
 actualParameter => [(actualParameterName => 'parm1', value => 'myval'),
 actualParameterName => 'parm2', value => 'val2')]}, ... => ...
```

In other words, the value of the `actualParameter` key in the optional arguments hash is a list of hashes, each representing one actual parameter. If the subprocedure call only takes one actual parameter, the value of the `actualParameter` key can be specified as just the hash representing the one parameter:

```perl
actualParameter => {actualParameterName => 'parm1', value => 'myval'}
```

**Example**

```
$cmdr->createStep("Test Proj", "Run Build", "Common Cleanup", {subprocedure => "Delay"},
```
actualParameter => {actualParameterName => 'Delay Time', value => '5'}});

ectool

**syntax:** ectool createStep <projectName> <procedureName> <stepName> ...

Specifying actual parameters in an ectool call is also different than specifying other arguments. Specify each key-value as an equal-sign delimited value:

ectool createStep ... --actualParameter "Delay Time=5" "parm2=val2"

**Note:** If the parameter name or value contains spaces, quotes are needed.

**Examples**

ectool createStep "Test Proj" "Run Build" "Compile" --command "make all"

ectool createStep "Test Proj" "Run Build" "Common Cleanup" --subprocedure "Delay" --actualParameter "Delay Time=5"

---

**deleteProcedure**

Deletes a procedure, including all steps.

You must specify a **projectName** and **procedureName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>The name of the procedure you want to delete.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains this procedure.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, procedureName

**Response**

None or a status OK message.

**ec-perl**

**syntax:** $cmdr->deleteProcedure(<projectName>, <procedureName>);

**Example**

$cmdr->deleteProcedure("Test Proj", "Run Build");

**ectool**

**syntax:** ectool deleteProcedure <projectName> <procedureName>

**Example**

ectool deleteProcedure "Test Proj" "Run Build"
**deleteStep**

Deletes a step from a procedure.

You must specify **projectName**, **procedureName**, and **stepName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>The name of the procedure that contains this step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains this procedure/step.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step you want to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**projectName**, **procedureName**, **stepName**

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `$cmdr->deleteStep(<projectName>, <procedureName>, <stepName>);`

*Example*

```
$cmdr->deleteStep("Test Proj", "Run Build", "Compile");
```

**ectool**

*Syntax:* `ectool deleteStep <projectName> <procedureName> <stepName>`

*Example*

```
ectool deleteStep "Test Proj" "Run Build" "Compile"
```

**getProcedure**

Finds a procedure by its name.

You must specify a **projectName** and a **procedureName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>The name of the procedure you are retrieving.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the procedure to retrieve.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**projectName**, **procedureName**
**Response**

One **procedure** element, which includes the procedure ID, name, time created, job name template, owner, resource name, workspace name, project name, and more.

**ec-perl**

*syntax:* $cmdr->getProcedure(<projectName>, <procedureName>);

*Example*

$cmdr->getProcedure("Test Proj", "Run Build");

**ectool**

*syntax:* ectool getProcedure <projectName> <procedureName>

*Example*

ectool getProcedure "Test Proj" "Run Build"

**getProcedures**

Retrieves all procedures in one project.

You must specify the **projectName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the procedures to retrieve.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`

**Response**

One or more **procedure** elements.

**ec-perl**

*syntax:* $cmdr->getProcedures(<projectName>);

*Example*

$cmdr->getProcedures("Test Proj");

**ectool**

*syntax:* ectool getProcedures <projectName>

*Example*

ectool getProcedures "Test Proj"

Back to Top
**getStep**

Retrieves a step from a procedure.

You must specify **projectName, procedureName, and stepName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>The name of the procedure that contains the step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project where you want to find a step.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, procedureName, stepName

**Response**

One step element.

**ec-perl**

*Syntax:* $cmdr->getStep(<projectName>, <procedureName>, <stepName>);

*Example*

```
$cmdr->getStep("Test Proj", "Run Build", "Compile");
```

**ectool**

*Syntax:*  ectool getStep <projectName> <procedureName> <stepName>

*Example*

```
ectool getStep "Test Proj" "Run Build" "Compile"
```

Back to Top

**getSteps**

Retrieves all steps in a procedure.

You must specify the **projectName and procedureName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>The name of the procedure that contains the steps.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the procedure for the steps you want to find.</td>
</tr>
</tbody>
</table>
Positional arguments
  projectName, procedureName

Response
  Zero or more step elements.

ec-perl
  syntax: $cmdr->getSteps(<projectName>, <procedureName>);

  Example
  $cmdr->getSteps("Test Proj", "Run Build");

ectool
  syntax: ectool getSteps <projectName> <procedureName>

  Example
  ectool getSteps "Test Proj" "Run Build"

modifyProcedure
  Modifies an existing procedure.

  You must specify projectName and procedureName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>jobNameTemplate</td>
<td>Job name format for jobs created by running this procedure.</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the procedure.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure to modify.</td>
</tr>
</tbody>
</table>
## API Commands - Procedure Management

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project to modify. Also requires procedureName</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the default resource where steps belonging to this procedure will run. This name may be a resource pool name.</td>
</tr>
<tr>
<td>timeLimit</td>
<td>If no time limit was specified on the calling step, time limits are copied to the calling step from the procedure. If the procedure is called from runProcedure (or a schedule), the time limit acts as a global job timeout. The &quot;timer&quot; for the procedure starts as soon as the calling step/job becomes runnable (all preconditions are satisfied).</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>Time limit units are hours</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the default workspace where job output is stored.</td>
</tr>
</tbody>
</table>

### Positional arguments

projectName, procedureName

### Response

None or a status OK message.

### ec-perl

**Syntax:**

```perl
$cmdr->modifyProcedure(<projectName>, <procedureName>, {...});
```

**Example**

```perl
$cmdr->modifyProcedure("Test Proj", "Run Build", {resourceName => "Windows - Bldg. 11");
```

### ectool

**Syntax:**

```bash
ectool modifyProcedure <projectName> <procedureName> ...
```

**Example**

```bash
ectool modifyProcedure "Test Proj" "Run Build"
    --resourceName "Windows - Bldg. 11"
```

### modifyStep

Modifies an existing step. You must specify projectName, procedureName, and stepName.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure.</td>
</tr>
<tr>
<td>alwaysRun</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>broadcast</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>command</td>
<td>The step command.</td>
</tr>
<tr>
<td>commandFile</td>
<td>This option is supported only in Perl and ectl tool bindings - it is not part of the XML protocol. The contents of the command file is read and stored in the “command” field. This is an alternative argument for command and is useful if the “command” field spans multiple lines.</td>
</tr>
<tr>
<td>condition</td>
<td>If empty or non-zero, the step will run. If set to &quot;0&quot;, the step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps.</td>
</tr>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms: relative (for example, “cred1”) - the credential is assumed to be in the project that contains the request target object. absolute (for example, “/projects/BuildProject/credentials/cred1”) - the credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
</tbody>
</table>
| description               | A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> `<pre> <span> <style> <table> `<tc> `<td> `<th> `<tr> `<ul>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| errorHandling | Determines what happens to the procedure if the step fails:  
|               |     * failProcedure - The current procedure continues, but the overall status is error (default).  
|               |     * abortProcedure - Aborts the current procedure, but allows already-running steps in the current procedure to complete.  
|               |     * abortProcedureNow - Aborts the current procedure and terminates running steps in the current procedure.  
|               |     * abortJob - Aborts the entire job, terminates running steps, but allows alwaysRun steps to run.  
|               |     * abortJobNow - Aborts the entire job and terminates all running steps, including alwaysRun steps.  
|               |     * ignore - Continues as if the step succeeded.  
| exclusive     | If set to 1, indicates this step should acquire and retain this resource exclusively. The value for exclusive is a `<Boolean flag -0|1|true|false>`. Defaults to "false". Note: Setting exclusive, sets exclusiveMode to "job".                                                                                                                                 |
| exclusiveMode | Use one of the following options:  
|               |     * None - the "default", which does not retain a resource.  
|               |     * Job - keeps the resource for the duration of the job. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" is used in a step. Future steps for this job will use this resource in preference to other resources--if this resource meets the needs of the steps and its step limit is not exceeded.  
|               |     * Step - keeps the resource for the duration of the step.  
|               |     * Call - keeps the resource for the duration of the procedure that called this step, which is equivalent to 'job' for top level steps.  
<p>| logFileName   | A custom log file name produced by running the step. By default, ElectricCommander assigns a unique name to this file.                                                                                                                                                         |
| newName       | Supply any name of your choice to rename the step.                                                                                                                                                                                                                           |
| parallel      | <code>&lt;Boolean flag - 0|1|true|false&gt;</code> Indicates if this step should run at the same time as adjacent steps marked to run as parallel also.                                                                                                                                                      |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>precondition</td>
<td>By default, if the step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a precondition is evaluated. A precondition is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE. Precondition example: Assume we defined these 4 steps: 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system Step 1 is an ordinary serial step. Steps 2 and 3 can run in parallel because they depend only on step 1's completion. Step 4 depends on step 2, but not step 3. You can achieve optimal step execution order with preconditions: • Make steps 2-4 run in parallel. • Step 2 needs a job property set at the end of its step to indicate step 2 is completing ([/myJob/buildInstallerCompleted=1]). • Set a precondition in step 4: $[/myJob/buildInstallerCompleted]</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the step to modify. <strong>Also requires</strong> projectName</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the step to modify. <strong>Also requires</strong> procedureName</td>
</tr>
<tr>
<td>postProcessor</td>
<td>The name of a program to run (script) after a step completes. This program looks at the step output to find errors and warnings. ElectricCommander includes a customizable program called &quot;postp&quot; for this purpose.</td>
</tr>
<tr>
<td>releaseExclusive</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>releaseMode</td>
<td>Use one of the following options:</td>
</tr>
<tr>
<td></td>
<td>• None - the &quot;default&quot; - no action if the resource was not previously marked as &quot;retain&quot;.</td>
</tr>
<tr>
<td></td>
<td>• Release - releases the resource at the end of this step. If the resource for the step was previously acquired with &quot;Retain exclusive&quot; (either by this step or some preceding step), the resource exclusivity is canceled at the end of this step. The resource is released in the normal way so it may be acquired by other jobs.</td>
</tr>
<tr>
<td></td>
<td>• Release to job - allows a step to promote a Step exclusive resource to a Job exclusive resource.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource used by this step.</td>
</tr>
<tr>
<td>shell</td>
<td>Where shell is the name of a program used to execute commands contained in the &quot;command&quot; field. The name of a temporary file containing commands will be appended to the end of this invocation line. Normally, this file is a command shell, but it could be any other command line program. The default is to use the standard shell for the platform it runs on.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step.</td>
</tr>
<tr>
<td></td>
<td>Also requires projectName and procedureName</td>
</tr>
<tr>
<td>subprocedure</td>
<td>The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the command or commandField.</td>
</tr>
<tr>
<td>subproject</td>
<td>If a subprocedure argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.</td>
</tr>
<tr>
<td>timeLimit</td>
<td>The maximum length of time the step is allowed to run. After the time specified, the step will be aborted.</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>&lt;hours</td>
</tr>
<tr>
<td>workingDirectory</td>
<td>The Commander agent sets this directory as the &quot;current working directory,&quot; running the command contained in the step. If no working directory is specified in the step, Commander uses the directory it created for the job in the Commander workspace.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace used by this step.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- projectName, procedureName, stepName
Response
None or a status OK message.

ec-perl

**syntax:** $cmdr->modifyStep(<projectName>, <procedureName>, <stepName>, ...
{<optionals>});

**Example**

$cmdr->modifyStep("Test Proj", "Run Build", "Compile", {commandFile => "tempfile.txt"});

ectool

**syntax:** ectool modifyStep <projectName> <procedureName> <stepName> ...

**Example**

ectool modifyStep "Test Proj" "Run Build" "Compile" --commandFile tempfile.txt

Back to Top

**moveStep**

Moves a step within a procedure.

You must specify **projectName, procedureName, and stepName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>beforeStep</td>
<td>Moves the step (stepName) to position before the step &quot;named&quot; by this option. If omitted, stepName is moved to the end of the list of steps.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the step to move.</td>
</tr>
</tbody>
</table>
| projectName   | The name of the project containing the step to move.  
**Also requires** procedureName |
| stepName      | The name of the step to move.  
**Also requires** projectName and procedureName |

Positional arguments
**projectName, procedureName, stepName**

Response
None or a status OK message.

ec-perl

**syntax:** $cmdr->moveStep(<projectName>, <procedureName>, <stepName>, (...<optionals>)});
**Example**

$cmdr->moveStep("Test Proj", "Run Build", "Get Sources", {beforeStep => "Compile"});

**ectool**

syntax: ectool moveStep <projectName> <procedureName> <stepName> ...

**Example**

ectool moveStep "Test Proj" "Run Build" "Get Sources"
--beforeStep "Compile"

Back to Top
API Commands - Process

createProcess

createProcess
Creates a new process for an application or component.

**Required Arguments**

projectName

- **Description**: Name of the project; must be unique among all projects.
- **Argument Type**: String

processName

- **Description**: Name of the process.
- **Argument Type**: String

**Optional Arguments**

applicationName

- **Description**: Name of the application, if the process is owned by an application; must be unique among all projects.
- **Argument Type**: String

cOMPONENTAPPLICATIONNAME

- **Description**: If specified, the component is scoped to this application, not the project.
- **Argument Type**: String

cOMPONENTNAME

- **Description**: Name of the component, if the process is owned by a component.
- **Argument Type**: String

cREDENTIALNAME

- **Description**: Name of a credential to attach to this process.
- **Argument Type**: String

dESCRIPTION
**Description**: Comment text describing this object; not interpreted at all by the ElectricCommander platform.

**Argument Type**: String

processType

**Description**: Defines the type of action performed by the process.

**Argument Type**: ProcessType

timeLimit

**Description**: Maximum amount of time that the step can execute; abort if it exceeds this time.

**Argument Type**: String

timeLimitUnits

**Description**: Units for the step-time limit: seconds, minutes, or hours.

**Argument Type**: TimeLimitUnits

workspaceName

**Description**: Name of the default workspace for this process.

**Argument Type**: String

**Response**

Returns a process component element.

e-perl

**Syntax**:

```perl
$<object>-&gt;createProcess(<projectName>, <processName>, {<optionals>});
```

**Example**:

```perl
$ec-&gt;createProcess("default", "process1", {componentName =&gt; "VCScomponent"});
```

ectool

**Syntax**:

```bash
ectool createProcess <projectName> <processName> [optionals...]
```

**Example**:

```bash
ectool createProcess default newProcess --componentName VCScomponent
```

**deleteProcess**

Deletes an application or component process.

**Required Arguments**

**projectName**

**Description**: Name of the project; must be unique among all projects.

**Argument Type**: String
processName

**Description:** Name of the process.

**Argument Type:** String

**Optional Arguments**

applicationName

**Description:** Name of the application, if the process is owned by an application; must be unique among all projects.

**Argument Type:** String

cOMPONENTAPPLICATIONNAME

**Description:** If specified, the component is scoped to this application, not the project.

**Argument Type:** String

componentName

**Description:** Name of the component, if the process is owned by a component.

**Argument Type:** String

**Response**

None or a status OK message.

e-perl

**Syntax:**

```perl
$self->deleteProcess($projectName, $processName, {$optional => 1});
```

**Example:**

```perl
$ec->deleteProcess("default", "newProcess", 
{componentName => "Component1"});
```

ectool

**Syntax:**

`ectool deleteProcess <projectName> <processName> [optionals...]`

**Example:**

`ectool deleteProcess default newProcess --componentName Component1`

**getProcess**

Retrieves an application or component process.

**Required Arguments**

projectName

**Description:** Name of the project; must be unique among all projects.

**Argument Type:** String

processName
**getProcess**

Retrieves all processes in an application or component.

**Required Arguments**

`projectName`

**Description:** Name of the project; must be unique among all projects.

**Argument Type:** String

**Optional Arguments**

`applicationName`

**Description:** Name of the application, if the process is owned by an application; must be unique among all projects.

**Argument Type:** String

`componentApplicationName`

**Description:** If specified, the component is scoped to this application, not the project.

**Argument Type:** String

`componentName`

**Description:** Name of the component, if the process is owned by a component.

**Argument Type:** String

**Response**

Retrieves the specified process element.

**ec-perl**

**Syntax:**

```
$<object>-getProcess(<projectName>, <processName>, {<optionals>});
```

**Example:**

```
$ec->getProcess("default", "newProcess", {componentName => "VCS"});
```

**ectool**

**Syntax:**

```
ectool getProcess <projectName> <processName> [optionals...]
```

**Example:**

```
ectool getProcess default newProcess --componentName VCScomponent
```
**Description:** Name of the application, if the process is owned by an application; must be unique among all projects.

**Argument Type:** String

**ComponentApplicationName**

**Description:** Application name of the component, if the component is scoped to application.

**Argument Type:** String

**ComponentName**

**Description:** Name of the component, if the process is owned by a component.

**Argument Type:** String

**Response**

Retrieves zero or more process elements.

**ec-perl**

**Syntax:**

```
<Object>-getProcesses(<projectName>, {<optionals>});
```

**Example:**

```
$ec->getProcesses("default", {componentName => "VCS"});
```

**ectool**

**Syntax:**

```
ectool getProcesses <projectName> [optionals...]
```

**Example:**

```
ectool getProcesses default --componentName VCScomponent
```

**modifyProcess**

Modifies an existing process.

**Required Arguments**

**projectName**

**Description:** Name of the project; must be unique among all projects.

**Argument Type:** String

**processName**

**Description:** Name of the process.

**Argument Type:** String

**Optional Arguments**

**applicationName**

**Description:** Name of the application, if the process is owned by an application; must be unique among all projects.
Argument Type: String
componentApplicationName

Description: If specified, the component is scoped to this application, not the project.
Argument Type: String
componentName

Description: Name of the component, if the process is owned by a component.
Argument Type: String
credentialName

Description: Name of a credential to attach to this process.
Argument Type: String
description

Description: Comment text describing this object; not interpreted at all by the ElectricCommander platform.
Argument Type: String
newName

Description: New name for an existing object that is being renamed.
Argument Type: String
processType

Description: Defines the type of action performed by the process.
Argument Type: ProcessType
timeLimit

Description: Maximum amount of time that the step can execute; abort if it exceeds this time.
Argument Type: String
timeLimitUnits

Description: Units for step time limit: seconds, minutes, or hours.
Argument Type: TimeLimitUnits
workspaceName

Description: Name of the default workspace for this process.
Argument Type: String

Response

Retrieves an updated process element.

ec-perl

Syntax:

$<object>-modifyProcess (<projectName>, <processName>, {<optionals>});
Example:

```
$ec->modifyProcess("default", "newProcess", (componentName => "VCS",
    newName => "VCScomponent", description => "An updated description"));
```

extool
Syntax:
```
extool modifyProcess <projectName> <processName> [optionals...]
```
Example:
```
extool modifyProcess default newProcess --componentName VCScomponent
    --newName VCS --description "A description"
```

runProcess

Runs the specified process.

**Required Arguments**

**projectName**

- **Description:** Name for the project; must be unique among all projects.
- **Argument Type:** String

**applicationName**

- **Description:** Name of the application that owns the process; must be unique among all applications in the project.
- **Argument Type:** String

**processName**

- **Description:** Name of the application process.
- **Argument Type:** String

**tierMapName**

- **Description:** Name of the tier map used to determine where to run the process.
- **Argument Type:** String

**Optional Arguments**

**actualParameter**

- **Description:** Parameters passed as arguments to the process.
- **Argument Type:** Map

**destinationProject**

- **Description:** Project that will own the job.
- **Argument Type:** String

**priority**
**Description:** Priority of the job.

**Argument Type:** JobPriority

**validate**

**Description:** Validates that the application process, tier map, and environment are well-defined and valid before the running the application process. This argument defaults to true.

**Argument Type:** Boolean

**Response**

Returns new job ID.

**ec-perl**

**Syntax:**

```
$<object>-&gt;runProcess(<projectName>, <applicationName>, <processName>,
                <tierMapName>, {[<optionals>]})
```

**Example:**

```
                 {destinationProject =&gt; "deploy1"});
```

**ectool**

**Syntax:**

```
ectool runProcess <projectName> <applicationName> <processName> <tierMapName>
[optionals...]
```

**Example:**

```
ectool runProcess default NewApp newProcess TierMap2 --destinationProject deploy1
```
API Commands - Process Dependency

createProcessDependency
createProcessDependency
createProcessDependency
createProcessDependency
createProcessDependency

createProcessDependency

createProcessDependency
Creates a dependency between two process steps.

Required Arguments

projectName

Description: Name of the project; must be unique among all projects.
Argument Type: String

processName

Description: Name of the process.
Argument Type: String

processStepName

Description: Name of the process step.
Argument Type: String

targetProcessStepName

Description: Name of the target process step.
Argument Type: String

Optional Arguments

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.
Argument Type: String

branchCondition

Description: Condition of the branch.
Argument Type: String

branchConditionName

Description: Name of the branch condition.
Argument Type: String

branchConditionType
Description: Type of the branch condition.

Argument Type: BranchConditionType

branchType

Description: Type of the branch.

Argument Type: BranchType

componentApplicationName

Description: If specified, the component is scoped to this application not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Returns a process dependency element.

del ProcessDependency

Deletes a dependency between two process steps.

Required Arguments

projectName

Description: Name of the project; must be unique among all projects.

Argument Type: String

processName
**Description**: Name of the process.

**Argument Type**: String

processStepName

**Description**: Name of the process step.

**Argument Type**: String

targetProcessStepName

**Description**: Name of the target process step.

**Argument Type**: String

**Optional Arguments**

applicationName

**Description**: Name of the application, if the process is owned by an application; must be unique among all projects.

**Argument Type**: String

componentApplicationName

**Description**: If specified, the component is scoped to this application, not the project.

**Argument Type**: String

componentName

**Description**: Name of the component, if the process is owned by a component.

**Argument Type**: String

**Response**

None or a status OK message.

**ec-perl**

**Syntax**:

```perl
$<object>-&gt;deleteProcessDependency(<projectName>, <processName>,
<processStepName>, <targetProcessStepName>, {<optionals>});
```

**Example**:

```perl
{componentName =&gt; "VCScomponent");
```

**ectool**

**Syntax**:

```bash
ectool deleteProcessDependency <projectName> <processName> <processStepName>
<targetProcessStepName> [optionals...]
```

**Example**:

```bash
ectool deleteProcessDependency default newProcess "Step B" "Step C"
--componentName VCScomponent
```
**getProcessDependencies**

Retrieves all dependencies for a process.

**Required Arguments**

- **projectName**
  - **Description:** Name of the project; must be unique among all projects.
  - **Argument Type:** String

- **processName**
  - **Description:** Name of the process.
  - **Argument Type:** String

**Optional Arguments**

- **applicationName**
  - **Description:** Name of the application, if the process is owned by an application; must be unique among all projects.
  - **Argument Type:** String

- **componentApplicationName**
  - **Description:** If specified, the component is scoped to this application, not the project.
  - **Argument Type:** String

- **componentName**
  - **Description:** Name of the component, if the process is owned by a component.
  - **Argument Type:** String

**Response**

Retrieves zero or more process dependency elements.

**ec-perl**

**Syntax:**

```perl
$<object>-getProcessDependencies(<projectName>, <processName>,
   [<optionals>]);
```

**Example:**

```perl
$ec->getProcessDependencies("default", "newProcess",
   {componentName => "VCScomponent");
```

**ectool**

**Syntax:**

```bash
ectool getProcessDependencies <projectName> <processName> [optionals...]
```

**Example:**

```bash
ectool getProcessDependencies default newProcess --componentName VCScomponent
```
modifyProcessDependency

Modifies a dependency between two process steps.

Required Arguments

projectName

**Description**: Name of the project; must be unique among all projects.

**Argument Type**: String

processName

**Description**: Name of the process.

**Argument Type**: String

processStepName

**Description**: Name of the process step.

**Argument Type**: String

targetProcessStepName

**Description**: Name of the target process step.

**Argument Type**: String

Optional Arguments

applicationName

**Description**: Name of the application, if the process is owned by an application.

**Argument Type**: String

branchCondition

**Description**: Condition of the branch.

**Argument Type**: String

branchConditionName

**Description**: Name of the branch condition.

**Argument Type**: String

branchConditionType

**Description**: Type of the branch condition.

**Argument Type**: BranchConditionType

branchType

**Description**: Type of the branch.

**Argument Type**: BranchType

cOMPONENTApplicationName
**Description:** If specified, the component is scoped to this application, not the project.

**Argument Type:** String

`componentName`

**Description:** Name of the component, if the process is owned by a component.

**Argument Type:** String

`ec-perl`

**Syntax:**

```perl
$<object>-modifyProcessDependency(<projectName>, <processName>, <processStep Name>, <targetProcessStepName>, {<optionals>});
```

**Example:**

```bash
$ec-modifyProcessDependency("default", "newProcess", "Step1", "StepA", componentName => "VCScomponent");
```

`ectool`

**Syntax:**

```bash
ectool modifyProcessDependency <projectName> <processName> <processStepName> <targetProcessStepName> [optionals...]
```

**Example:**

```bash
ectool modifyProcessDependency default newProcess Step1 StepA --componentName VCScomponent
```
API Commands - Process Step

createProcessStep
deleteProcessStep
getProcessStep
getProcessSteps
modifyProcessStep

Note: Several of the following API commands contain context type optional arguments. For example, a step command may reference either a procedure or component.

createProcessStep

Creates a new process step.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

processStepName

Description: Name of the process step.

Argument Type: String

Optional Arguments

actualParameters

Description: Actual parameters (<var1>=<val1> [<var2>=<val2> ...]) passed to an invoked subprocedure or process.

Argument Type: Map

afterProcessStep

Description: If specified, the process step will be placed after the named process step.

Argument Type: String

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String
applicationTierName

**Description:** Application tier on which to run the step.
**Argument Type:** String

beforeProcessStep

**Description:** If specified, the process step will be placed before the named process step.
**Argument Type:** String

compName

**Description:** Name of the component, if the process is owned by a component.
**Argument Type:** String

credentialName

**Description:** Name of the credential object.
**Argument Type:** String

description

**Description:** Comment text describing this object; not interpreted at all by the ElectricCommander platform.
**Argument Type:** String

errorHandling

**Description:** Specifies error handling for this step.
**Argument Type:** ErrorHandling

includeCompParameterRef

**Description:** True if the actual parameters should be generated from component properties. Works for artifact components only.
**Argument Type:** Boolean

processStepType

**Description:** Defines type of the process step.
**Argument Type:** ProcessStepType

subcomponent

**Description:** If referencing a component process, the name of the component.
**Argument Type:** String

subcomponentProcess

**Description:** If referencing a component process, the name of the component process.
**Argument Type:** String

subprocedure

**Description:** If referencing a procedure, the name of the procedure.
Argument Type: String

subproject

Description: If referencing a procedure, the name of the procedure's project.

Argument Type: String
timeLimit

Description: Maximum amount of time that the step can execute; abort if it exceeds this time.

Argument Type: String
timeLimitUnits

Description: Units for the step time limit: seconds, minutes, or hours.

Argument Type: TimeLimitUnits
workspaceName

Description: Name of the workspace.

Argument Type: String

Response

Returns a process step element.

ec-perl

Syntax:

```perl
$<object>->createProcessStep(<projectName>, <processName>, <processStepName>, {<optionals>});
```

Example:

```perl
$ec->createProcessStep("default", "newProcess", "Step 1",
    {componentName => "VCScomponent"});
```

ectool

Syntax:

```bash
ectool createProcessStep <projectName> <processName> <processStepName>
    [optionals...]
```

Example:

```bash
ectool createProcessStep default newProcess "Step A"
    --componentName VCScomponent
```

deleteProcessStep

Deletes an application or component process step.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.
**Argument Type:** String

processName

**Description:** Name of the process.

**Argument Type:** String

processStepName

**Description:** Name of the process step.

**Argument Type:** String

Optional Arguments

**Argument Type:** String

applicationName

**Description:** Name of the application, if the process is owned by an application; must be unique among all projects.

**Argument Type:** String

componentApplicationName

**Description:** If specified, the component is scoped to this application, not the project.

**Argument Type:** String

componentName

**Description:** Name of the component, if the process is owned by a component.

**Argument Type:** String

Response

None or a status OK message.

**ec-perl**

**Syntax:**

```
<object>-deleteProcessStep (<projectName>, <processName>,
<processStepName>, {<optionals>});
```

**Example:**

```
$ec->deleteProcessStep ("default", "newProcess", "stepToDelete",
(componentName=> "VCScomponent"));
```

**ectool**

**Syntax:**

```
ectool deleteProcessStep <projectName> <processName> <processStepName>
[optionals...]
```

**Example:**

```
ectool deleteProcessStep default newProcess "stepToDelete"
--componentName VCScomponent
```

**getProcessStep**
Gets an application or component process step.

**Required Arguments**

**projectName**
- **Description**: Name for the project; must be unique among all projects.
- **Argument Type**: String

**processName**
- **Description**: The name of the process.
- **Argument Type**: String

**processStepName**
- **Description**: The name of the process step.
- **Argument Type**: String

**Optional Arguments**

**applicationName**
- **Description**: Name of the application, if the process is owned by an application; must be unique among all projects.
- **Argument Type**: String

**componentApplicationName**
- **Description**: If specified, the component is scoped to this application, not the project.
- **Argument Type**: String

**componentName**
- **Description**: Name of the component, if the process is owned by a component.
- **Argument Type**: String

**Response**

Retrieves the specified process step element.

**ec-perl**
- **Syntax**:
  ```perl
  $<object>-getProcessStep(<projectName>, <processName>, <processStepName>,
  {[<optionals>]})
  
  Example:
  $ec->getProcessStep("default", "newProcess", "Step 1",
  {componentName => "VCScomponent"});
  ```

**ectool**
- **Syntax**:
  ```bash
  ectool getProcessStep <projectName> <processName> <processStepName>
  [optionals...]
  
  Example:
  ```
getProcessSteps

Retrieves all the process steps in an application or component process.

**Required Arguments**

**projectName**

*Description:* Name for the project; must be unique among all projects.

*Argument Type:* String

**processName**

*Description:* Name of the process.

*Argument Type:* String

**Optional Arguments**

**applicationName**

*Description:* Name of the application, if the process is owned by an application; must be unique among all projects.

*Argument Type:* String

**componentApplicationName**

*Description:* If specified, the component is scoped to this application, not the project.

*Argument Type:* String

**componentName**

*Description:* Name of the component, if the process is owned by a component.

*Argument Type:* String

**Response**

Retrieves zero or more process step elements.

**ec-perl**

*Syntax:*

$$\langle\text{object}\rangle->\text{getProcessSteps}\langle<\text{projectName}>\text{, }<\text{processName}>\text{, }\{<\text{optionals}>\}\rangle;$$

*Example:*

```perl
$ec->getProcessSteps("default", "newProcess", 
                    \{componentName=> "VCScomponent"\});
```

**ectool**

*Syntax:*

```bash
ectool getProcessSteps <projectName> <processName> [optionals...]
```

*Example:*
modifyProcessStep

Modifies an existing process step.

Required Arguments

projectName

Description: Name of the project; must be unique among all projects.

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

processStepName

Description: Name of the process step.

Argument Type: String

Optional Arguments

actualParameters

Description: Actual parameters passed to an invoked subprocedure or process.

Argument Type: Map

afterProcessStep

Description: If specified, the process step will be placed after the named process step.

Argument Type: String

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

applicationTierName

Description: Name of the application tier on which to run the step.

Argument Type: String

beforeProcessStep

Description: If specified, the process step will be placed before the named process step.

Argument Type: String

clearActualParameters

Description: True if the step should remove all actual parameters.
Argument Type: Boolean

componentApplicationName

Description: If specified, the component is scoped to this application not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

credentialName

Description: Name of the credential object.

Argument Type: String

description

Description: Comment text describing this object; not interpreted at all by the ElectricCommander platform.

Argument Type: String

errorHandling

Description: Specifies error handling for this step.

Argument Type: ErrorHandling

includeCompParameterRef

Description: True if the actual parameters should be generated from component properties. Works for artifact components only.

Argument Type: Boolean

newName

Description: New name for an existing object that is being renamed.

Argument Type: String

processStepType

Description: Defines type of the process step.

Argument Type: ProcessStepType

subcomponent

Description: If referencing a component process, the name of the component.

Argument Type: String

subcomponentApplicationName

Description: If referencing a component process, the name of the component application (if it has not been scoped to a project).

Argument Type: String
subcomponentProcess

**Description:** If referencing a component process, the name of the component process.

**Argument Type:** String

subprocedure

**Description:** If referencing a procedure, the name of the procedure.

**Argument Type:** String

subproject

**Description:** If referencing a procedure, the name of the procedure's project.

**Argument Type:** String

timeLimit

**Description:** Maximum amount of time that the step can execute; abort if it exceeds this time.

**Argument Type:** String

timeLimitUnits

**Description:** Units for the step time limit: seconds, minutes, or hours.

**Argument Type:** TimeLimitUnits

workspaceName

**Description:** Name of the workspace.

**Argument Type:** String

**Response**

Retrieves an updated process step element.

ec-perl

**Syntax:**

```
$obj->modifyProcessStep(<projectName>, <processName>,
<processStepName>, {[optionals]});
```

**Example:**

```
$ec->modifyProcessStep("default", "newProcess", "Step 1",
{componentName => "VCScomponent", newName => "Step 2",
description => "A description");
```

ectool

**Syntax:**

```
ectool modifyProcessStep <projectName> <processName> <processStepName>
[optionals...]
```

**Example:**

```
ectool modify ProcessStep newProcess "Step A"
--componentName VCScomponent --newName "Step B"
--description "A description"
```
createProject

Creates a new project.

You must specify a projectName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| credentialName   | credentialName can be one of two forms: 
|                  | relative 
|                  | (for example, "cred1") - the credential is assumed to be in the 
|                  | project that contains the request target object. 
|                  | absolute 
|                  | (for example, "/projects/BuildProject/credentials/cred1") - the 
|                  | credential can be from any specified project, regardless of the 
|                  | target object's project. |
| description      | A plain text or HTML description for this object. 
|                  | If using HTML, you must surround your text with 
|                  | `<html> ... </html>` tags. The only HTML tags allowed in the 
|                  | text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> 
|                  | <p> <pre> <span> <style> <table> <tc> <td> <th> 
|                  | <tr> <ul>` |
| projectName      | This is any name of your choice for your new project. |
| resourceName     | The name of the resource to use as the default for steps run by 
|                  | procedures in this project. |
| workspaceName    | The name of a workspace to use as the default for steps run by 
|                  | procedures in this project. |

Positional arguments

`projectName`

Response

None or a status OK message.

ec-perl

`syntax`: `cmdr->createProject(<projectName>, {<optionals>});`
Example

```php
$cmdr->createProject(“Test Proj”, {workspaceName => ”Test_WS”});
```

dectool

**syntax:**

dectool createProject <projectName> ...

**Example**

dectool createProject ”Test Proj” --workspaceName ”Test WS”

deleteProject

Deletes a project, including all procedures, procedure steps, and jobs within that project.

You must specify a **projectName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project you want to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- **projectName**

**Response**

None or a status OK message.

ec-perl

**syntax:**

```perl
$cmdr->deleteProject(<projectName>);
```

**Example**

$cmdr->deleteProject(“Test Proj”);

dectool

**syntax:**
dectool deleteProject <projectName>

**Example**

dectool deleteProject "Test Proj"

getProject

Finds a project by its name.

You must specify a **projectName**.
Arguments | Descriptions
---|---
projectName | The name of the project you need to retrieve.

**Positional arguments**

```
projectName
```

**Response**

One `project` element.

**ec-perl**

*Syntax:* `cmdr-getProject(<projectName>);

*Example*

```
$cmdr-getProject("Test Proj");
```

**ectool**

*Syntax:* `ectool getProject <projectName>`

*Example*

```
ectool getProject "Test Proj"
```

**getProjects**

Retrieves all projects.

Arguments | Descriptions
---|---
None | None

**Positional arguments**

None

**Response**

Zero or more `project` elements.

*Note:* This response includes all projects in the system, including plugin projects, which are not displayed on the Projects page in the web UI.

**ec-perl**

*Syntax:* `cmdr-getProjects();`

*Example*

```
$cmdr-getProjects();
```
### modifyProject

Modifies an existing project.

You must specify a `projectName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| **credentialName** | `credentialName` can be one of two forms:  
relative  
(for example, "cred1") - the credential is assumed to be in the  
project that contains the request target object.  
absolute  
(for example, "/projects(BuildProject)/credentials/cred1") - the  
credential can be from any specified project, regardless of the  
target object's project. |
| **description** | A plain text or HTML description for this object.  
If using HTML, you must surround your text with  
<html> ... </html> tags. The only HTML tags allowed in the  
text are: <a> <b> <br> <div> <dl> <font> <i> <li> <ol>  
<p> <pre> <span> <style> <table> <tc> <td> <th>  
<tr> <ul> |
| **newName** | Supply any name of your choice to rename the project. |
| **projectName** | The name of the project you want to modify. |
| **resourceName** | The name of the resource used as the default for steps run by  
procedures in this project. |
| **workspaceName** | The name of the default workspace where job output is stored. |

#### Positional arguments

`projectName`

#### Response

None or a status OK message.

#### ec-perl

```
syntax: $cmdr->modifyProject(<projectName>, {...});
```
Example

```php
$cmdr->modifyProject("Test Proj", \{description => "A very simple project"\});
```

**ectool**

**syntax:** `ectool modifyProject <projectName> ...`

**Example**

`ectool modifyProject "Test Proj" --description "A very simple project"

Back to Top
API Commands - Property Management

createProperty
deleteProperty
evalScript
expandString
getProperties
getProperty
incrementProperty
modifyProperty
setProperty

createProperty

Creates a regular string or nested property sheet using a combination of property path and context.

You must specify a propertyName and locator arguments to define where (or on which object) you are creating this property.

**Note:** The name "properties" is NOT a valid property name.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>The name of the property to create. It may be a relative or absolute property path, including &quot;my&quot; paths such as &quot;/myProject/prop1&quot;.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential container of the property sheet which owns the property.</td>
</tr>
<tr>
<td></td>
<td>credentialName can be one of two forms:</td>
</tr>
<tr>
<td></td>
<td>relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object.</td>
</tr>
<tr>
<td></td>
<td>If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: a b br div font i li ol p pre span style table tc td th tr ul</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment container of the property sheet which owns the property; must be unique among all projects</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>The name of the environment tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>expandable</td>
<td>Whether or not the property is recursively expandable.</td>
</tr>
<tr>
<td></td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>extendedContextSearch</td>
<td>For simple property names, whether or not to search objects in the hierarchy to find the desired property.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group where you want to create a property.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by findObjects and getObjects.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin where you want to create a property.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure. <strong>Must be combined with its projectName.</strong></td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>propertyType</td>
<td>`&lt;string</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource where you want to define the new property.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step. If you are using a step name to define the location for the new property, you must use projectName and procedureName also.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule. If you're using a schedule name to define the location for the new property, you must use projectName also.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>The name of the special system object. In this context, only server is legal.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
</tbody>
</table>
## API Commands - Property Management

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The user name where you want to add a property.</td>
</tr>
<tr>
<td>value</td>
<td>For a string property (see <code>propertyType</code> above), this is the value of the property. For a sheet property, this argument is invalid.</td>
</tr>
<tr>
<td>valueFile</td>
<td>This option is supported only in Perl and ectool bindings - it is not a part of the XML protocol. The contents of the <code>valueFile</code> is read and stored in the &quot;value&quot; field for a string property. This is an alternative argument for <code>value</code> and is useful if the &quot;value&quot; field spans multiple lines.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace where you want to add a property.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

### Positional arguments

- `propertyName`

### Response

An XML stream that echoes the new property, including its ID, which is assigned by the ElectricCommander server.

#### ec-perl

**syntax:** `$cmdr->createProperty(<propertyName>, {<optionals>});`

**Examples**

```perl
$cmdr->createProperty('/myJob/Runtime Env/PATH', {value => 'c:\bin'});
$cmdr->createProperty('Runtime Env/PATH', {value => 'c:\bin', ...});
```

#### ectool

**syntax:** `ectool createProperty <propertyName> ...`

**Examples**

```bash
ectool createProperty "/myJob/Runtime Env/PATH" --value "c:\bin"
ectool createProperty "Runtime Env/PATH" --value "c:\bin" --jobId 4fa765dd-73f1-11e3-b67e-b0a420524153
```

Back to Top
**deleteProperty**

Deletes a property from a property sheet.

You must specify a `propertyName` and you must specify locator arguments to find the property you want to delete.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>propertyName</code></td>
<td>The name of the property to delete.</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td><code>applicationTierName</code></td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>artifactName</code></td>
<td>The name of the artifact container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>artifactVersionName</code></td>
<td>The name of the artifact version. Note: An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td><code>componentName</code></td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>configName</code></td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td><code>credentialName</code></td>
<td>Whether or not the property is recursively expandable.</td>
</tr>
<tr>
<td></td>
<td>credentialName can be one of two forms:</td>
</tr>
<tr>
<td></td>
<td>relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td><code>environmentName</code></td>
<td>The name of the environment container of the property sheet which owns the property; must be unique among all projects</td>
</tr>
<tr>
<td><code>environmentTierName</code></td>
<td>The name of the environment tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| extendedContextSearch| For simple property names, whether or not to search objects in the hierarchy to find the desired property.  
<Boolean flag -0|1|true|false>  If set, and there is an object specifier in the command, ElectricCommander first looks for the property in that object specifier, but also searches in other locations if not found, according to the following rules:  
1. If the object specifier is a procedure, ElectricCommander looks for the property in the project where the procedure resides.  
2. If the object specifier is a job step, Commander looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties.  
Default setting is "true." |
| gatewayName          | The name of the gateway.                                                                                                                   |
| groupName            | The name of a group that contains this property.                                                                                           |
| jobId                | The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template. |
| jobStepId            | The unique identifier for a job step, assigned automatically when the job step is created.                                                 |
| notifierName         | The name of the email notifier.                                                                                                             |
| objectId             | This is an object identifier returned by findObjects and getObjects.  
pluginName           | The name of a plugin that may contain a property you want to delete.  
procedureName        | The name of the procedure containing the property you want to delete.  
Also requires projectName |
| processName          | The name of the process, if the container is a process or process step.                                                                    |
| processStepName      | The name of the process step, if the container is a process step.                                                                        |
| projectName          | The name of the project that contains the property you want to delete.                                                                     |
| propertySheetId      | The unique identifier for a property sheet, assigned automatically when the property sheet is created.                                   |
| repositoryName       | The name of the repository for artifact management.                                                                                         |
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>The name of the resource that contains the property you want to delete.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing the property you want to delete.</td>
</tr>
<tr>
<td></td>
<td>Also requires <code>projectName</code></td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step containing the property you want to delete.</td>
</tr>
<tr>
<td></td>
<td>Also requires <code>projectName</code> and <code>procedureName</code></td>
</tr>
<tr>
<td>systemObjectName</td>
<td>The name of a special system object. Only 'sever' is legal in this context.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The user name that contains this property.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace containing this property.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

### Positional arguments

- `propertyName`

### Response

None or a status OK message.

### ec-perl

**syntax:**

```perl
$cmdr->deleteProperty(<propertyName>, [ ... ]);  
```

**Example**

```perl
$cmdr->deleteProperty("/projects/Sample project/Changeset ID");
```

### ectool

**syntax:**

```
ectool deleteProperty <propertyName> ...
```
**Example**

ectool deleteProperty "'/projects/Sample project/Changeset ID"

**evalScript**

Evaluates a script in a given context. This API is similar to `expandString` except that it evaluates the `value` argument as a Javascript block, without performing any property substitution on either the script or the result. The string value of the final expression in the script is returned as the `value` element of the response.

You must specify a `value` to evaluate.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The script to evaluate.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential container of the property sheet which owns the property. <code>credentialName</code> can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>The name of the environment tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of a group where you might evaluate a script.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by findObjects and getObjects.</td>
</tr>
<tr>
<td>path</td>
<td>Property path string.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of a plugin where you might evaluate a script.</td>
</tr>
</tbody>
</table>
| procedureName      | The name of a procedure where you might need to evaluate a script.  
**Also requires** projectName                                                                 |
| projectName        | The name of the process, if the container is a process or process step.                                                                        |
| processStepName    | The name of the process step, if the container is a process step.                                                                              |
| projectName        | The name of the project that contains the script to evaluate.  
**Also requires** projectName                                                                 |
| propertySheetId    | The unique identifier for a property sheet, assigned automatically when the property sheet is created.                                            |
| repositoryName     | The name of the repository for artifact management.                                                                                           |
| resourceName       | The name of a resource where you might evaluate a script.                                                                                     |
| resourcePoolName   | The name of a pool containing one or more resources.                                                                                           |
| scheduleName       | The name of a schedule within this project.  
**Also requires** projectName                                                                 |
<p>| stateDefinitionName| The name of the state definition.                                                                                                               |
| stateName          | The name of the state.                                                                                                                         |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepName</td>
<td>The name of the step whose script you might evaluate. Also requires projectName and procedureName</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>System object names include: admin</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user where you may need to evaluate a script.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of a workspace where you may need to evaluate a script.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

**Positional arguments**

value

**Response**

The string value of the final expression in the Javascript block inside a value element.

**ec-perl**

**syntax:** $cmdr->evalScript (<value>);

**Examples**

my $result = $ec->evalScript (q("ip" + server.hostIP +", name=" + server.hostName) ) ->findvalue("/value");

my $result = $ec->evalScript (q{myProject.projectName}, {jobId => '4fa765dd-73f1-11e3-b67e-b0a420524153'});

**ectool**

**syntax:** ectool evalScript <value>

**Examples**

ectool evalScript "'ip" + server.hostIP +", name=" + server.hostName'

ectool evalScript 'myProject.projectName' --jobId 4fa765dd-73f1-11e3-b67e-b0a420524153
    --jobStepId 5da765dd-73f1-11e3-b67e-b0a420524153
**expandString**

Expands property references in a string, in the current context.

You must specify a `value` and a context in which to perform the expansion or a `valueFile` option.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The string value to expand in the given context.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact.</td>
</tr>
</tbody>
</table>
| artifactVersionName | The name of the artifact version. 
  **Note:** An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name—the Commander server interprets either name form correctly. |
| componentName    | The name of the component container of the property sheet which owns the property.                                                             |
| configName      | The name of the emailConfig container that owns the property.                                                                               |
| credentialName  | The name of the credential container of the property sheet which owns the property. 
  **credentialName** can be one of two forms: 
  **relative** (for example, "cred1") - the credential is assumed to be in the project that contains the request target object. 
  **absolute** (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project. |
<p>| environmentName | The name of the environment container of the property sheet which owns the property; must be unique among all projects.                       |
| environmentTierName | The name of the environment tier container of the property sheet which owns the property.                                                      |
| gatewayName     | The name of the gateway.                                                                                                                     |
| groupName       | The name of a group where you might expand a string.                                                                                         |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</td>
</tr>
<tr>
<td>path</td>
<td>Property path string.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of a plugin where you might expand a string.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of a procedure where you might need to expand a string. <strong>Also requires</strong> projectName</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project that contains the string to expand.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of a resource where you might expand a string.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of a schedule within this project. <strong>Also requires</strong> projectName</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step whose string you might be expanding. <strong>Also requires</strong> projectName and procedureName</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>System object names include: admin</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user where you may need to expand the string.</td>
</tr>
<tr>
<td>valueFile</td>
<td>This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the valueFile is read and stored in the &quot;value&quot; field. This is an alternative argument for value and is useful if the value field spans multiple lines.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of a workspace where you may need to expand the string.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

### Positional arguments

- **value**

  The expanded string value.

### Response

- **value**

  The expanded string value.

### ec-perl

**syntax:** $cmdr->expandString(<value>, {<optionals>});

**Examples**

```perl
$cmdr->expandString('$$[fullUserName]', {userName => "admin"})->findvalue('/value') ->value();
$cmdr->expandString('$$[/myWorkspace/agentUncPath]/$$[logFileName]',
  {jobStepId => 5da765dd-73f1-11e3-b67e-b0420524153})->findvalue('/value')->value();
```

### ectool

**syntax:** ectool expandString <value> ...

**Examples**

```bash
ectool expandString '$[fullUserName]' --userName admin
ectool expandString '$[/myWorkspace/agentUncPath]/$[logFileName]'
  --jobStepId 5da765dd-73f1-11e3-b67e-b0420524153
```

Back to Top
**getProperties**

Retrieves all properties associated with an object, along with the property sheet identifier for the object's property sheet.

You must specify object locators for the properties you want to retrieve.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential containing the properties to retrieve. <strong>credentialName</strong> can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project. Also requires <strong>projectName</strong></td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>The name of the environment tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>expand</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group containing the properties to retrieve.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</td>
</tr>
<tr>
<td>path</td>
<td>The path to the property sheet containing the properties to retrieve. If the full path to the property sheet is specified, no additional object locators are needed.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin containing the properties to retrieve.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the properties to retrieve. <strong>Also requires</strong> <code>projectName</code></td>
</tr>
<tr>
<td>procName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>procStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the properties to retrieve.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>recurse</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource containing the properties to retrieve.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing the properties to retrieve. <strong>Also requires</strong> <code>projectName</code></td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
stateDefinitionName | The name of the state definition.
stateName | The name of the state.
stepName | The name of the step containing the properties to retrieve. 
**Also requires** projectName and procedureName
systemObjectName | The name of the system object containing the properties to retrieve. Only "server" is supported.
transitionDefinitionName | The name of the transition definition.
transitionName | The name of the transition.
userName | The name of the user containing the properties to retrieve.
workflowDefinitionName | The name of the workflow definition.
workflowName | The name of the workflow. 
**Also requires** projectName
workspaceName | The name of the workspace containing the properties to retrieve.
zoneName | The name of the zone.

**Positional arguments**
Arguments to locate the property, beginning with the top-level object.

**Response**
A propertySheet element, which contains zero or more property elements and nested propertySheet elements.

**ec-perl**
- **syntax:** $cmdr-&gt;getProperties({<optionals>});

**Examples**
$cmdr-&gt;getProperties({resourceName =&gt; "r2"});

**ectool**
- **syntax:** ectool getProperties ...

**Examples**
ectool getProperties --resourceName "r2"

Back to Top
**getProperty**

Retrieves the specified property value.

You must specify a `propertyName`.

**Note:** This specification can be the full path to the property or it can be relative to an object, which then requires appropriate object locators.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>propertyName</code></td>
<td>The name or path for the property to retrieve.</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td><code>applicationTierName</code></td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>artifactName</code></td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td><code>artifactVersionName</code></td>
<td>The name of the artifact version. Note: An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td><code>componentName</code></td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>configName</code></td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td><code>credentialName</code></td>
<td>The name of the credential containing the property to retrieve. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project. Also requires <code>projectName</code></td>
</tr>
<tr>
<td><code>environmentName</code></td>
<td>The name of the environment container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td><code>environmentTierName</code></td>
<td>The name of the environment tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>expand</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>
| extendedContextSearch | For simple property names, whether or not to search objects in the hierarchy to find the desired property.  
If set, and there is an object locator in the command, Commander first looks for the property in that object locator, but also searches in other locations if not found, according to the following rules:  
If the object locator is a procedure, Commander looks for the property in the project where the procedure resides.  
If the object locator is a job step, Commander looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties.  
Default setting is "true."                                                                 |
| gatewayName        | The name of the gateway.                                                                                                                                                                                          |
| groupName          | The name of the group containing the property to retrieve.                                                                                                                                                       |
| jobId              | The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.                                 |
| jobStepId          | The unique identifier for a job step, assigned automatically when the job step is created.                                                                                                                       |
| notifierName       | The name of the email notifier.                                                                                                                                                                                     |
| objectId           | This is an object identifier returned by `findObjects` and `getObjects`.                                                                                                                                            |
| pluginName         | The name of the plugin containing the property to retrieve.                                                                                                                                                       |
| procedureName      | The name of the procedure containing the property to retrieve.  
**Also requires** `projectName`                                                                                                                                 |
<p>| projectName        | The name of the process, if the container is a process or process step.                                                                                                                                               |
| processStepName    | The name of the process step, if the container is a process step.                                                                                                                                                   |
| projectName        | The name of the project containing the property to retrieve.                                                                                                                                                       |
| propertySheetId    | The unique identifier for a property sheet, assigned automatically when the property sheet is created.                                                                                                           |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource containing the property to retrieve.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing the property to retrieve.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step containing the property to retrieve.</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>The name of the system object containing the property to retrieve.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user containing the property to retrieve.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace containing the property to retrieve.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

**Positional arguments**

propertyName

**Response**

A property sheet or a text string containing the value of the property.

Property value example: 35491

**ec-perl**

*Syntax:* `cmdr->getProperty(<propertyName>, {<optionals>});`

**Examples**

use XML::XPath;
`cmdr->getProperty("/myProject/changeset ID")->findvalue('//value')->{'value'};`
$cmdr->getProperty("Changeset ID", [projectName => "Sample Project"])\->\findvalue('/\value')\->\value();

**ectool**

**syntax:** ectool getProperty <propertyName> ...

**Examples**

ectool getProperty "/myProject/changeset ID"

ectool getProperty "Changeset ID" --projectName "Sample Project"

# Retrieve the /users/<userName>/providerName property.

ectool getProperty --objectID <ID> --propertyName "/users/<userName>/providerName"

---

**incrementProperty**

Atomically increments the specified property value by the `incrementBy` amount. If the property does not exist, it will be created with an initial value of the `incrementBy` amount.

You must specify a `propertyName` and `incrementBy`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>The name of the property to increment.</td>
</tr>
<tr>
<td>incrementBy</td>
<td>This is positive or negative integer.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
</tbody>
</table>

Back to Top
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>The name of the environment tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>extendedContextSearch</td>
<td>For simple property names, whether or not to search objects in the hierarchy to find the desired property. &lt;Boolean flag&gt; 0</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group containing the property to increment.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by findObjects and getObjects.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin containing a property to increment.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing this property. Also requires projectName.</td>
</tr>
</tbody>
</table>
## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>processName</code></td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td><code>processStepName</code></td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td><code>projectName</code></td>
<td>The name of the project containing this property.</td>
</tr>
<tr>
<td><code>propertySheetId</code></td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td><code>repositoryName</code></td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td><code>resourceName</code></td>
<td>The name of the resource containing this property.</td>
</tr>
<tr>
<td><code>resourcePoolName</code></td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td><code>scheduleName</code></td>
<td>The name of the schedule containing this property.</td>
</tr>
<tr>
<td><code>stateDefinitionName</code></td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td><code>stateName</code></td>
<td>The name of the state.</td>
</tr>
<tr>
<td><code>stepName</code></td>
<td>The name of the step containing this property.</td>
</tr>
<tr>
<td><code>systemObjectName</code></td>
<td>Only server is a valid system object for this API.</td>
</tr>
<tr>
<td><code>transitionDefinitionName</code></td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td><code>transitionName</code></td>
<td>The name of the transition.</td>
</tr>
<tr>
<td><code>userName</code></td>
<td>The name of the user containing this property.</td>
</tr>
<tr>
<td><code>workflowDefinitionName</code></td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td><code>workflowName</code></td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td><code>workspaceName</code></td>
<td>The name of the workspace containing this property.</td>
</tr>
<tr>
<td><code>zoneName</code></td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

### Positional arguments

- `propertyName, incrementBy`

### Response

A text string containing the updated numeric property value.
modifyProperty

Modifies a regular string or nested property sheet using a combination of property path and context.

You must specify a propertyName.

Note: The name "properties" is NOT a valid property name.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>The name of the property to be modified; must be unique within the property sheet. This argument can be a path.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
</tbody>
</table>
| credentialName     | **credentialName** can be one of two forms:  
  **relative**  
  (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.  
  **absolute**  
  (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.                                                                                                                    |
| description         | A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a>`, `<b>`, `<br>`, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, `<ul>` |
| environmentName    | The name of the environment container of the property sheet which owns the property; must be unique among all projects.                                                                                                                                                                      |
| environmentTierName| The name of the environment tier container of the property sheet which owns the property.                                                                                                                                                                                                                                                   |
| expandable         | `<Boolean flag -0|1|true|false>` - Determines whether the property value will undergo property expansion when it is fetched. Default is "true".                                                                                                               |
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>extendedContextSearch</td>
<td>For simple property names, whether or not to search objects in the hierarchy to find the desired property. If set, and there is an object specified in the command, ElectricCommander first looks for the property in that object specifier, but also searches in other locations if not found, according to the following rules: 1) If the object specifier is a procedure, ElectricCommander looks for the property in the project where the procedure resides. 2) If the object specifier is a job step, ElectricCommander looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. Default setting is &quot;true.&quot;</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group containing the property to be modified.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the property.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin containing the property to be modified.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the property to be modified. <strong>Also requires</strong> <code>projectName</code></td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the property to be modified. Note that the property may be on the project itself or on a contained object, indicated by other arguments.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>propertyType</td>
<td>`&lt;string</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource containing the property to be modified.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing the property to be modified.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step containing the property to be modified.</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>System objects include: admin</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user containing the property to be modified.</td>
</tr>
<tr>
<td>value</td>
<td>This can be any string you choose to add to a property.</td>
</tr>
<tr>
<td>valueFile</td>
<td>This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. The contents of the value file is read and stored in the &quot;value&quot; field. This is an alternative argument for value and is useful if the &quot;value&quot; field spans multiple lines.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace containing the property to be modified.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

**Positional arguments**

propertyName
Response
An XML stream that echoes the modified property.

ec-perl

*Syntax:* $cmdr->modifyProperty(<propertyName>, {...});

*Example*

$cmdr->modifyProperty("Saved Variables", {description =>
    "Starting configuration of name/value pairs", jobId => 4fa765dd-73f1-11e3-b67
e-b0a420524153});

ectool

*Syntax:* ectool modifyProperty <propertyName> ...

*Example*

ectool modifyProperty "Saved Variables" --description "Starting configuration of name/value pairs" --jobId 4fa765dd-73f1-11e3-b67e-b0a420524153

setProperty

Sets the value for the specified property.

You must specify a *propertyName* and *value*. The property name can be the full path to the property or it can be relative to an object, which then means you must use object locators to specify the property.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>The name or path of the property you want to set; must be unique within the property sheet.</td>
</tr>
<tr>
<td></td>
<td>This argument can be a path.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the property.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential containing the property you want to set. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project. Also requires projectName</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tt&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>The name of the environment tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>expandable</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>extendedContextSearch</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>1) If the object specified is a procedure, ElectricCommander looks for the property in the project where the procedure resides.</td>
</tr>
<tr>
<td></td>
<td>2) If the object specified is a job step, Commander looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties.</td>
</tr>
<tr>
<td></td>
<td>Default setting is &quot;false.&quot;</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group containing the property you want to set.</td>
</tr>
<tr>
<td>jobId</td>
<td>The name of the job containing the property you want to set. The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The name of the job step containing the property you want to set. The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>objectId</td>
<td>This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin containing the property you want to set.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the property you want to set.</td>
</tr>
<tr>
<td></td>
<td>Also requires <code>projectName</code></td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the property you want to set.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>The name of the resource containing the property you want to set.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule containing the property you want to set. <strong>Also requires</strong> projectName</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step containing the property you want to set. <strong>Also requires</strong> projectName and procedureName</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>The name of the system object containing the property you want to set.</td>
</tr>
<tr>
<td></td>
<td>System objects include: admin</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user containing the property you want to set.</td>
</tr>
<tr>
<td>valueFile</td>
<td>This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the valuefile is read and stored in the &quot;value&quot; field. This is an alternative argument for value and is useful if the value field spans multiple lines.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace containing the property you want to set.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

### Positional arguments

**propertyName, value**

### Response

An XML stream that echoes the property.

**ec-perl**

**syntax:** $cmdr->setProperty(<propertyName>, <value>, {<optionals>});
**Examples**

$cmdr->setProperty("Changeset ID", "14992", [projectName => "Sample Project"]);

$cmdr->setProperty("/myResource/Application Path", "c:\Program Files\Application");

$cmdr->setProperty("Application Path", "c:\Program Files\Application",
  [resourceName => "r2"]);

**ectool**

*syntax:*  ectool setProperty <propertyName> <value> ...

**Examples**

ectool setProperty "Changeset ID" "14992" --projectName "Sample Project"

ectool setProperty "/myResource/Application Path" "c:\Program Files\Application"

ectool setProperty "Application Path" "c:\Program Files\Application"
  --resourceName "r2"

[Back to Top](#)
API Commands - Resource Management

addResourcesToPool
addResourceToEnvironmentTier
createResource
createResourcePool
deleteResource
deleteResourcePool
getResource
getResources
getResourcesInEnvironmentTier
getResourcesInPool
getResourcePool
getResourcePools
getResourceUsage
modifyResource
pingAllResources
pingResource
removeResourceFromEnvironmentTier
removeResourcesFromPool

addResourcesToPool

Adds resources to a specific resource pool. A resource pool is a named group of resources.

You must specify a resourcePoolName and one or more resource names.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceNames</td>
<td>The list of resources to add to the pool.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
</tbody>
</table>

Positional arguments
resourcePoolName, resourceName(s)
Response

None or status OK message.

**ec-perl**

*Syntax:* `$cmdr->addResourcesToPool(<resourcePoolName>, {resourceName => [...]});`

*Example*

```perl
$cmdr->addResourcesToPool("pool1", { resourceName => ["resource1", "resource2", "resource3"]});
```

**ectool**

*Syntax:* `ectool addResourcesToPool <resourcePoolName> --resourceNames <resourceName1> ...
(Note the plural form for the resourceNames option)`

*Example*

```bash
ectool addResourcesToPool "Test Pool" --resourceNames Test1 Test2 Test3
```

Back to Top

**addResourceToEnvironmentTier**

Adds the given resource to the given environment tier.

You must specify the `resourceName, projectName, environmentName, and environmentTierName` arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resourceName</code></td>
<td>Name for the resource; must be unique among all resources.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>projectName</code></td>
<td>Name for the project; must be unique among all projects; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>environmentName</code></td>
<td>Name of the environment; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>environmentTierName</code></td>
<td>Name for the environment tier; must be unique among all tiers for the environment.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

None or a status OK message.
ec-perl

Syntax:

$<object>-&gt;addResourceToEnvironmentTier(<resourceName>, <projectName>, <environmentName>, <environmentTierName>);

Example:

$ec-&gt;addResourceToEnvironmentTier("Resource1", "default", "newEnv", "envTier1");

ectool

Syntax:

addResourceToEnvironmentTier <resourceName> <projectName> <environmentName> <environmentTierName>

Example:

ectool addResourceToEnvironmentTier Resource1 default newEnv envTier1

createResource

Creates a new resource.

Important Note: For a proxy resource, proxyHostName and proxyPort arguments refer to the proxying Commander agent.
hostName and port refer to the proxy target.

You must specify a resourceName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactCacheDirectory</td>
<td>The directory on the agent host where retrieved artifacts are stored.</td>
</tr>
<tr>
<td>block</td>
<td>&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>
| description             | A plain text or HTML description for this object. If using HTML, you must surround your text with 
<p>|                         | &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt; |
| hostName                | The name or IP address of the computer containing the ElectricCommander agent for this resource if it's an ordinary resource. If this is a proxy resource, this is the name or IP address of the proxy target. |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pools</td>
<td>A space-separated list of one or more pool names where this resource is a member. Steps defined to run on a resource pool will run on any available member (resource) in the pool.</td>
</tr>
<tr>
<td>port</td>
<td>The Commander agent port number for an ordinary resource. If a port number is not specified, the default agent port is used. The default agent port can be configured on the Server Settings page in the Commander Web Interface. For a proxy resource, this is the port number for the service running on the proxy target that will run commands on behalf of the Commander agent. For ssh, the default is 22.</td>
</tr>
<tr>
<td>proxyCustomization</td>
<td>Perl code customizing how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources.</td>
</tr>
<tr>
<td>proxyHostName</td>
<td>The name or IP address of the computer containing the Commander Agent used for a proxy resource.</td>
</tr>
<tr>
<td>proxyPort</td>
<td>The Commander agent port number for a proxy resource. See the port argument description for more details.</td>
</tr>
<tr>
<td>proxyProtocol</td>
<td>Protocol for communicating with the proxy target. Defaults to ssh. (This argument is not exposed in the Commander Web Interface at this time.)</td>
</tr>
<tr>
<td>resourceDisabled</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>repositoryNames</td>
<td>A list of one or more repository names—each repository name listed on a &quot;new line&quot;.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the new resource you are creating.</td>
</tr>
<tr>
<td>shell</td>
<td>This sets a default shell for running step commands on this resource. The default is &quot;cmd /q /c&quot; for a Windows agent and &quot;sh -e&quot; for a UNIX agent.</td>
</tr>
<tr>
<td>stepLimit</td>
<td>Limits the number of steps that can run on the resource at one time. Setting the limit to 1 enforces serial access to the resource.</td>
</tr>
</tbody>
</table>
**API Commands - Resource Management**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| trusted     | `<Boolean flag - 0|1|true|false>` If "true", the resource is trusted. A trusted agent is one that has been "certificate verified." Agents can be either trusted or untrusted:  
- trusted - the Commander server verifies the agent's identity using SSL certificate verification.  
- untrusted - the Commander server does not verify agent identity. Potentially, an untrusted agent is a security risk. |
| useSSL      | `<Boolean flag - 0|1|true|false>` Use this flag to define whether or not SSL is used for server-agent communication, or if you need to use SSL to communicate with your Active Directory servers. Default is "true". |
| workspaceName| The name of the workspace this resource will use.                                                                                           |
| zoneName    | The name of the zone where this resource resides.                                                                                           |

**Positional arguments**

resourceName

**Response**

None or a status OK message.

**ec-perl**

`syntax:` $cmdr->createResource(<resourceName>, {<optionals>});

**Example**

```perl
$cmdr->createResource("Test Resource 1", {hostName => "localhost", pools => "P1 P 2"});
```

**ectool**

`syntax:` ectool createResource <resourceName> ...

**Example**

```bash
ectool createResource "Test Resource 1" --hostName localhost --pools "P1 P2"
```

**createResourcePool**

Creates a new pool for resources.

You must specify a `resourcePoolName`. 
### Arguments

<table>
<thead>
<tr>
<th></th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoDelete</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code></td>
</tr>
<tr>
<td>resourceNames</td>
<td>A list of resource names to add to the pool. This value does not need to refer to an existing resource. Any names that do not resolve to an existing resource will be skipped when assigning resources to steps.</td>
</tr>
<tr>
<td>orderingFilter</td>
<td>A Javascript block invoked when scheduling resources for a pool. <strong>Note:</strong> A Javascript block is not required unless you need to override the default resource ordering behavior.</td>
</tr>
<tr>
<td>resourcePoolDisabled</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>Choose any unique name for your resource pool.</td>
</tr>
</tbody>
</table>

### Positional arguments

**resourcePoolName**

### Response

Returns a `resourcePool` object.

**ec-perl**

**syntax:** `$cmdr->createResourcePool(<resourcePoolName>, {<optionals>});`  

**Example**

$cmdr->createResourcePool ("aPool", {resourceName => ["resource1", "resource2"]});

**ectool**

**syntax:** `ectool createResourcePool <resourcePoolName> ...`  

**Example**

`ectool createResourcePool aPool --resourceNames resource1 resource2`

### deleteResource

Deletes a resource.
You must supply a resourceName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>The name of the resource to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

resourceName

**Response**

None or a status OK message.

**ec-perl**

* syntax:*

```perl
$cmdr->deleteResource(<resourceName>);
```

* Example*

```perl
$cmdr->deleteResource("Test Resource 1");
```

**ectool**

* syntax:*

```bash
ectool deleteResource <resourceName>
```

* Example*

```bash
ectool deleteResource "Test Resource 1"
```

**deleteResourcePool**

Deletes a resource pool.

You must supply a resourcePoolName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool (containing one or more resources) to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

resourcePoolName

**Response**

None or a status OK message.

**ec-perl**

* syntax:*

```perl
$cmdr->deleteResourcePool(<resourcePoolName>);
```

* Example*

```perl
$cmdr->deleteResourcePool("Test Resource 1");
```
### ectool

**Syntax:**
```
ectool deleteResourcePool <resourcePoolName>
```

**Example**
```
ectool deleteResourcePool "Test Resource 1"
```

### getRESOURCE

Retrieves a resource by its name.

You must specify `resourceName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>The name of the resource to retrieve.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`resourceName`

**Response**

One resource element, which includes the resource ID, name, agent state, time created, host name, owner, port, disabled flag, shell, step limit, workspace name, and more. If using zones and gateways, `getResource` returns a list of gateways where this resource participates.

### ec-perl

**Syntax:**
```
$cmdr->getResource(<resourceName>);
```

**Example**
```
$cmdr->getResource("Test Resource 1");
```

### ectool

**Syntax:**
```
ectool getResource <resourceName>
```

**Example**
```
ectool getResource "Test Resource 1"
```

### getRESOURCES

Retrieves all resources.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
**Positional arguments**

None

**Response**

Zero or more *resource* elements.

**ec-perl**

syntax: $cmdr->getResources();

*Example*

$cmdr->getResources();

**ectool**

syntax: ectool getResources

*Example*

ectool getResources

**getResourcesInEnvironmentTier**

Returns the list of resources in an environment tier.

You must specify the *projectName, environmentName*, and *environmentTierName* arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projectName</strong></td>
<td>Name for the project; must be unique among all projects; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><strong>environmentName</strong></td>
<td>Name of the environment; must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><strong>environmentTierName</strong></td>
<td>Name for the environment tier; must be unique among all tiers for the environment.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

Retrieves zero or more resource elements in the specified environment tier.

**ec-perl**

Syntax:

```perl
$<object>-getResourcesInEnvironmentTier(<projectName>, <environmentName>, <environmentTierName>);
```
Example:

```
ec->getResourcesInEnvironmentTier("default", "newEnv", "envTier1");
```

tool syntax:
```
getResourcesInEnvironmentTier <projectName> <environmentName> <environmentTierName>
```

Example:
```
ec tool getResourcesInEnvironmentTier default newEnv envTier1
```

### getResourcesInPool

Retrieves a list of resources in a pool.

You must specify a pool (name).

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobStepId</td>
<td>The ID number of the job step related to this pool.</td>
</tr>
<tr>
<td>pool</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- pool

**Response**

An XML stream containing zero or more resource elements.

### ec-perl

**syntax:**
```
$cmdr->getResourcesInPool(<pool>);
```

**Example**
```
$cmdr->getResourcesInPool("WindowsPool");
```

### ectool

**syntax:**
```
ec tool getResourcesInPool <pool>
```

**Example**
```
ec tool getResourcesInPool WindowsPool
```

[Back to Top](#)
**getResourcePool**

Retrieves a specified resource pool by name.

You must specify a `resourcePoolName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resourcePoolName</code></td>
<td>The name of a pool containing one or more resources.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`resourcePoolName`

**Response**

An XML stream containing one `resourcePool` element.

**ec-perl**

*Syntax:* `$cmdr->getResourcePool(<resourcePoolName>);`

*Example*

```perl
$cmdr->getResourcePool("WindowsPool");
```

**ectool**

*Syntax:* `ectool getResourcePool <resourcePoolName>`

*Example*

```bash
ectool getResourcePool WindowsPool
```

**getResourceManager**

Retrieves a list of resource pools.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

An XML stream containing zero or more `resourcePool` elements.

**ec-perl**

*Syntax:* `$cmdr->getResourcePools;`
**Example**

```bash
$cmdr->getResourcePools;
```

**ectool**

**syntax:** `ectool getResourcePools`

**Example**

```bash
ectool getResourcePools
```

**getForResourceUsage**

Retrieves resource usage information.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

An XML stream containing zero or more `resourceUsage` elements.

**ec-perl**

**syntax:** `$cmdr->getResourceUsage;`

**Example**

```perl
$cmdr->getResourceUsage;
```

**ectool**

**syntax:** `ectool getResourceUsage`

**Example**

```bash
ectool getResourceUsage
```

**modifyResource**

Modifies an existing resource.

You must specify a `resourceName`.

**Important note:** For a proxy resource, `proxyHostName` and `proxyPort` arguments refer to the proxying Commander agent. `hostName` and `port` refer to the proxy target.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactCacheDirectory</td>
<td>The directory on the agent host where retrieved artifacts are stored.</td>
</tr>
<tr>
<td>block</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: `&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>hostName</td>
<td>The name or IP address for the ElectricCommander machine containing the agent for this resource.</td>
</tr>
<tr>
<td>newName</td>
<td>Supply any name of your choice to rename the resource.</td>
</tr>
<tr>
<td>pools</td>
<td>A space-separated list of one or more pool names where this resource is a member. The pool name can be used in place of a single resource name. ElectricCommander chooses a resource from the pool when it executes the job step.</td>
</tr>
<tr>
<td>port</td>
<td>The port number for the ElectricCommander agent. Default is to the default agent port, but you can change this port number because of port conflicts or multiple agents running on the same machine.</td>
</tr>
<tr>
<td>proxyCustomization</td>
<td>Perl code customizing how the proxy resource communicates with the proxy target. Only applicable for proxy resources.</td>
</tr>
<tr>
<td>proxyHostName</td>
<td>The IP address of the computer containing the ElectricCommander Agent used for a proxy resource.</td>
</tr>
<tr>
<td>proxyPort</td>
<td>The Commander agent port number for a proxy resource. See the port argument for more details.</td>
</tr>
<tr>
<td>proxyProtocol</td>
<td>Protocol for communicating with the proxy target. Defaults to ssh. This argument is not exposed in the Commander web interface at this time.</td>
</tr>
<tr>
<td>repositoryNames</td>
<td>A list of repository names with each repository name listed on a &quot;new line&quot;.</td>
</tr>
<tr>
<td>resourceDisabled</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource being modified.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>shell</td>
<td>This sets a default shell for running step commands on this resource. The default is &quot;cmd /q /c&quot; for a Windows agent and &quot;sh -e&quot; for a UNIX agent.</td>
</tr>
<tr>
<td>stepLimit</td>
<td>This limits the number of steps that can be running on the resource at one time. Setting this value to &quot;1&quot; is a good way to enforce serial access to the resource.</td>
</tr>
<tr>
<td>trusted</td>
<td>A trusted agent is one that has been &quot;certificate verified.&quot; Agents can be either trusted or untrusted:</td>
</tr>
<tr>
<td></td>
<td>- trusted - the Commander server verifies the agent's identity using SSL certificate verification.</td>
</tr>
<tr>
<td></td>
<td>- untrusted - the Commander server does not verify agent identity. Potentially, an untrusted agent is a security risk.</td>
</tr>
<tr>
<td>useSSL</td>
<td>Use this flag to define whether or not SSL is used for server-agent communication, or if you need to use SSL to communicate with your Active Directory servers. Default is &quot;true&quot;.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the default workspace where job output is stored.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone where this resource resides.</td>
</tr>
</tbody>
</table>

### Positional arguments

- resourceName

### Response

None or a status OK message.

### ec-perl

**syntax:** `$cmdr->modifyResource(<resourceName>, {...});`  

**Example**  

```perl  
$cmdr->modifyResource("Test Resource 1", {stepLimit => 5, shell => "bash"});  
```

### ectool

**syntax:** `ectool modifyResource <resourceName> ...`  

**Example**  

```ectool  
ectool modifyResource "Test Resource 1" --stepLimit 5 --shell "bash"  
```
**modifyResourcePool**

Modifies an existing resource pool.

You must specify a `resourcePoolName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoDelete</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>newName</td>
<td>Any new unique name you choose to rename this resource pool.</td>
</tr>
<tr>
<td>resourceNames</td>
<td>A list of resource names to add to the pool. This value does not need to refer to an existing resource. Any names that do not resolve to an existing resource will be skipped when assigning resources to steps.</td>
</tr>
<tr>
<td>orderingFilter</td>
<td>A Javascript block invoked when scheduling resources for a pool. Note: A Javascript block is not required unless you need to override the default resource ordering behavior.</td>
</tr>
<tr>
<td>resourcePoolDisabled</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `resourcePoolName`

**Response**

The modified `resourcePool` object.

**ec-perl**

- **syntax:** `cmdr->modifyResourcePool(<resourcePoolName>, {<optionals>});`

- **Example**
  ```perl
  cmdr->modifyResourcePool("WindowsPool", { resourcePoolDisabled => 1});
  ```

**ectool**

- **syntax:** `ectool modifyResourcePool <resourcePoolName> ...`
**Example**

ectool modifyResourcePool WindowsPool --resourcePoolDisabled 1

---

**pingAllResources**

Pings all resources.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

None or a status OK message.

**ec-perl**

syntax: $cmdr->pingAllResources({<optionals>});

Example

$cmdr->pingAllResources();

**ectool**

syntax: ectool pingAllResources...

Example

ectool pingAllResources

---

**pingResource**

Pings one resources.

You must specify a resourceName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>The name of the resource to ping.</td>
</tr>
</tbody>
</table>
**Positional arguments**

resourceName

**Response**

None or a status OK message.

c**ec-perl**

```perl
_syntax: $cmdr->pingResource($resourceName);

*Example*

$cmdr->pingResource("Test Resource 1");
```

c**ectool**

```bash
_syntax: ectool pingResource <resourceName> ...

*Example*

ectool pingResource "Test Resource 1"
```

**removeResourceFromEnvironmentTier**

Removes the given resource from the given environment tier.

You must specify the **resourceName**, **projectName**, **environmentName**, and **environmentTierName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name for the resource; must be unique among all resources. Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects; must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>Name of the environment; must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>Name for the environment tier; must be unique among all tiers for the environment. Argument Type: String</td>
</tr>
</tbody>
</table>

**Response**

None or a status OK message.
**ec-perl**

Syntax:

```perl
$object->removeResourceFromEnvironmentTier(<resourceName>, <projectName>, <environmentName>, <environmentTierName>);
```

Example:

```perl
$object->removeResourceFromEnvironmentTier("Resource1", "default", "newEnv", "envTier1");
```

**ectool**

Syntax:

```bash
removeResourceFromEnvironmentTier <resourceName> <projectName> <environmentName> <environmentTierName>
```

Example:

```bash
ectool removeResourceFromEnvironmentTier Resource1 default newEnv envTier1
```

**Back to Top**

---

**removeResourcesFromPool**

Removes resources from a specified resource pool.

You must specify a **resourcePoolName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceNames</td>
<td>The list of resources to remove from this pool.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
</tbody>
</table>

**Positional arguments**

resourcePoolName

**Response**

None or a status OK message.

**ec-perl**

**syntax:**

```perl
$cmdr->removeResourcesFromPool(<resourcePoolName>, {<optionals>});
```

**Example**

```perl
$cmdr->removeResourcesFromPool("Test Pool", {resourceNames => ["Test1", "Test2", "Test3"]});
```

**ectool**

**syntax:**

```bash
ectool removeResourcesFromPool <resourcePoolName> ...
```
**Example**

`ectool removeResourcesFromPool "Test Pool" --resourceNames Test1 Test2 Test3`
API Commands - Schedule Management

(createSchedule
deleteSchedule
getSchedule
getSchedules
modifySchedule

createSchedule

Creates a new schedule.

Note: If both startTime and stopTime are specified, intervalUnits and interval are used to specify an interval time to repeat running the procedure.

You must specify a projectName and scheduleName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure.</td>
</tr>
<tr>
<td>beginDate</td>
<td>&lt;yyyy-mm-dd&gt; The date you want the schedule to begin.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential to use for user impersonation when running the procedure. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>endDate</td>
<td>&lt;yyyy-mm-dd&gt; The date you want this schedule to end.</td>
</tr>
<tr>
<td>interval</td>
<td>Determines the repeat interval for starting new jobs.</td>
</tr>
</tbody>
</table>
## API Commands - Schedule Management

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| **intervalUnits** | Specifies the units for the interval argument  
<hours|minutes|seconds|continuous>  
If set to continuous, Commander creates a new job as soon as the previous job completes. |
| **misfirePolicy** |  
<ignore|runOnce>  
Specifies the misfire policy. A schedule may not fire at the allotted time because a prior job is still running, the server is running low on resources and there is a delay, or the server is down. When the underlying issue is resolved, the server will schedule the next job at the next regularly scheduled time slot if the policy is 'ignore', otherwise it will run the job immediately.  
Defaults to "ignore". |
| **monthDays** | Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space. |
| **priority** |  
<low|normal|high|highest>  
Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number.  
If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number. |
| **procedureName** | The procedure to run when the schedule is invoked. |
| **projectName** | The name of the project that contains the procedure this schedule will run. |
| **scheduleDisabled** |  
<Boolean flag - 0|1|true|false>  
If set to 1, Commander will not start any new jobs from the schedule. Defaults to "false". |
| **scheduleName** | This is any name of your choice for this schedule. |
| **startTime** | Enter hours and minutes, formatted hh:mm, using the 24-hour clock. Using this schedule, ElectricCommander starts creating jobs at this time on the specified days. |
| **stopTime** | Enter hours and minutes, formatted hh:mm, using the 24-hour clock. Commander stops creating new jobs at this time, but a job in progress will continue to run.  
If stopTime is not specified, ElectricCommander creates one job only on each specified day. |
| **timeZone** | Supply the time zone (string) you want to use for this schedule. |
| **weekDays** | Restricts the schedule to specified days of the week. Specify days of the week separated by spaces. Use English names "Monday", "Tuesday", and so on. |
Positional arguments

- projectName, scheduleName

Response

None or status OK message.

ec-perl

**syntax:** $cmdr->createSchedule(<projectName>, <scheduleName>, {<optionals>});

**Example**

```perl
$cmdr->createSchedule('Sample Project', 'Weekend', {startTime => '00:00', stopTime => '23:59', weekDays => 'Saturday Sunday', interval => 1, intervalUnits => 'hours', actualParameter => [{actualParameterName => 'param1', value => 'value1'}] });
```

ectool

**syntax:** ectool createSchedule <projectName> <scheduleName> ...

**Example**

```bash
ectool createSchedule "Sample Project" "Weekend" --start00:00 --stopTime 23:59 --weekDays "Saturday Sunday" --interval 1 --intervalUnits hours
```

deleteSchedule

Deletes a schedule.

You must specify a **projectName** and **scheduleName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The schedule you want to delete belongs to this project.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule you want to delete.</td>
</tr>
</tbody>
</table>

Positional arguments

- projectName, scheduleName

Response

None or a status OK message.

ec-perl

**syntax:** $cmdr->deleteSchedule(<projectName>, <scheduleName>);

**Example**

```perl
$cmdr->deleteSchedule("Sample Project","Weekend");
```
ectool

**Syntax:**
```
ectool deleteSchedule <projectName> <scheduleName>
```

**Example**
```
ectool deleteSchedule "Sample Project" "Weekend"
```

getSchedule

Retrieves a schedule by its name.

You must specify a `projectName` and `scheduleName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project that contains the schedule to retrieve.</td>
</tr>
<tr>
<td><code>scheduleName</code></td>
<td>The name of the schedule to retrieve.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, scheduleName`

**Response**

One `schedule` element.

ec-perl

**Syntax:**
```
$cmdr->getSchedule(<projectName>, <scheduleName>);
```

**Example**
```
$cmdr->getSchedule("Sample Project", "Build Schedule");
```

ectool

**Syntax:**
```
ectool getSchedule <projectName> <scheduleName>
```

**Example**
```
ectool getSchedule "Sample Project" "Build Schedule"
```

getSchedules

Retrieves all schedules.

You must specify a `projectName`.
### modifySchedule

Modifies an existing schedule.

You must specify a `projectName` and a `scheduleName`.

**Note:** If both `startTime` and `stopTime` are specified, `intervalUnits` and `interval` are used to specify an interval to repeat running the procedure.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure.</td>
</tr>
<tr>
<td>beginDate</td>
<td><code>&lt;yyyy-mm-dd&gt;</code> The date you want the schedule to begin.</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| credentialName | The name of the credential to use for user impersonation when running the procedure.  
|               | credentialName can be one of two forms:  
|               | relative  
|               | (for example, "cred1") - the credential is assumed to be in the project that contains the request target object.  
|               | absolute  
|               | (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project. |
| description   | A plain text or HTML description for this object. If using HTML, you must surround your text with  
|               | <html> ... </html> tags. The only HTML tags allowed in the text are: a, b, br, div, dl, font, i, li, ol, p, pre, span, style, table, tc, td, th, tr, ul. |
| endDate       | <yyyy-mm-dd> The date you want this schedule to end.                                                                                       |
| interval      | Determines the repeat interval for starting new jobs.                                                                                      |
| intervalUnits | Specifies the units for the interval argument  
|               | <hours|minutes|seconds|continuous>. If set to continuous, Commander creates a new job as soon as the previous job completes. |
| misfirePolicy | <ignore|runOnce> Specifies the misfire policy. A schedule may not fire at the allotted time because a prior job is still running, the server is running low on resources and there is a delay, or the server is down.  
|               | When the underlying issue is resolved, the server will schedule the next job at the next regularly scheduled time slot if the policy is 'Ignore', otherwise it will run the job immediately.  
|               | Defaults to 'Ignore'.                                                                                                                       |
| monthDays     | Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space.                |
| newName       | Supply any name of your choice to rename the schedule.                                                                                      |
| priority      | <low|normal|high|highest>  
|               | Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number. |
| procedureName | The name of the procedure to run when the schedule is invoked.                                                                           |
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the schedule to modify.</td>
</tr>
<tr>
<td>scheduleDisabled</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule to modify.</td>
</tr>
<tr>
<td>startTime</td>
<td>Enter hours and minutes, formatted <code>hh:mm</code>, using the 24-hour clock. Commander starts creating jobs at this time on the days specified.</td>
</tr>
<tr>
<td>stopTime</td>
<td>Enter hours and minutes, formatted <code>hh:mm</code>, using the 24-hour clock. Commander stops creating new jobs at this time, but a job in progress will continue to run. If <code>stopTime</code> is not specified, Commander creates one job only on each specified day.</td>
</tr>
<tr>
<td>timeZone</td>
<td>Supply the time zone you want to use for this schedule.</td>
</tr>
<tr>
<td>weekDays</td>
<td>Restricts the schedule to specified days of the week. Specify days of the week separated by spaces. Use English names &quot;Monday&quot;, &quot;Tuesday&quot;, and so on.</td>
</tr>
</tbody>
</table>

### Positional arguments

- `projectName`, `scheduleName`

### Response

- None or a status OK message.

### ec-perl

**syntax:**

```perl
$cmdr->modifySchedule(<projectName>, <scheduleName>, {...});
```

**Example**

```perl
$cmdr->modifySchedule("Sample Project", "Weekend",
  {procedureName => "Delay",
   actualParameter => {actualParameterName => "Delay Time",
                      value => "5"}});
```

### ectool

**syntax:**

```bash
ectool modifySchedule <projectName> <scheduleName> ...
```

**Example**

```bash
ectool modifySchedule "Sample Project" "Weekend" --procedureName "Delay"
--actualParameter "Delay Time=5"
```

---

**Back to Top**
API Commands - Server Management

getVersions

Retrieves server version information.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

Positional arguments

None

Response

A serverVersion element.

ec-perl

syntax: $cmdr->getVersions();

Example

$cmdr->getVersions();

ectool

syntax: ectool getVersions

Example

ectool getVersions

shutdownServer

Shuts down the ElectricCommander server. Shutting down the server can take as long as a couple of minutes, depending on the server activity level at the time the shutdown command is issued.

The Commander server is composed of two processes. The main process is a Java Virtual Machine (JVM). The second process, called the "wrapper", is responsible for interacting with the native operating system as a service. This wrapper process is responsible for starting and stopping the main JVM process.
### Arguments and Descriptions

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td><em>Boolean</em> flag - 0</td>
</tr>
<tr>
<td>restart</td>
<td><em>Boolean</em> flag - 0</td>
</tr>
</tbody>
</table>

### Positional arguments

None

### Response

None or a status OK message.

### ec-perl

**syntax:** $cmdr-&gt;shutdownServer({<optionals>});

**Example**

$cmdr-&gt;shutdownServer({restart => 1});

### ectool

**syntax:** ectool shutdownServer ...

**Example**

ectool shutdownServer --restart 1

---

### importLicenseData

Imports one or more licenses.

You must specify licenseData.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>licenseData</td>
<td>The content of a license file (perl</td>
</tr>
<tr>
<td>licenseFile</td>
<td><em>&lt;localFileName&gt;</em> The license file to import. This is a local file that will be read by ectool. The contents is sent as the licenseData argument (ectool only).</td>
</tr>
</tbody>
</table>

### Positional arguments

licenseData
**Response**

None or a status OK message.

**ec-perl**

*Syntax:* $cmdr->importLicenseData(<licenseData>)

*Example*

```perl
my $data = 'cat license.xml';
$cmdr->importLicenseData ($data);
```

**ectool**

*Syntax:* ectool importLicenseData <licenseData>

*Example*

```bash
ectool importLicenseData license.xml
```

**getAdminLicense**

Retrieves the admin license, which can be used when all concurrent user licenses are in use.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

You can receive one or more responses, depending on how you are licensed and actual license usage at the time of your query.

**Response examples:**

When the user does not have the necessary permission to use the Administrator license:

```xml
<error requestId="1">
  <code>AccessDenied</code>
  <where/>
  <message>Principal 'bob@company.com' does not have execute privileges on systemObject[name=licensing,id=10]</message>
  <details/>
</error>
```

When the user has permission to get/use the Administrator license, but already has a User license:

```xml
<result>User 'bob@company.com@192.168.17.217' already has an active license.</result>
```

When the user has permission to use/get the Administrator license, has no other license, and the Administrator license is not currently assigned:
<result>User 'bob@company.com@192.168.17.217' was given the admin license.</result>

When the user has permission to get/use the Administrator license, has no license, and the Administrator license is currently assigned to someone else:

<result>User 'joedoe@company.com@192.168.17.217' was given the admin license that previously belonged to 'bob@company.com@192.168.17.217'. </result>

e-c-perl

Syntax: $cmdr->getAdminLicense();

Example

$cmdr->getAdminLicense();

ectool

Syntax: ectool getAdminLicense

Example

ectool getAdminLicense

Back to Top

getLicense

Retrieves information for one license.

You must specify the productName and featureName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureName</td>
<td>The name of the licensed feature. Possible features include: Server</td>
</tr>
<tr>
<td>productName</td>
<td>The name of the product with the licensed feature. Possible products include: ElectricCommander</td>
</tr>
</tbody>
</table>

Positional arguments

productName, featureName

Response

One license element.

e-c-perl

Syntax: $cmdr->getLicense(<productName>, <featureName>);

Example

$cmdr->getLicense('ElectricCommander', 'Server');
ectool

**syntax:** `ectool getLicense <productName> <featureName>`

*Example*

`ectool getLicense ElectricCommander Server`

## getLicenses

Retrieves all license data.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Zero or more `license` elements.

**ec-perl**

**syntax:** `$cmdr->getLicenses();`

*Example*

`$cmdr->getLicenses();`

**ectool**

**syntax:** `ectool getLicenses`

*Example*

`ectool getLicenses`

## getLicenseUsage

Retrieves the current license usage.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

**Positional arguments**

None
Response

You may receive one or more responses for licenseUsage, depending on how you are licensed and actual license usage at the time of your query.

ec-perl

syntax: $cmdr->getLicenseUsage();

Example

$cmdr->getLicenseUsage();

ectool

syntax: ectool getLicenseUsage

Example

ectool getLicenseUsage

deleteLicense

Deletes a license.

You must specify a productName and featureName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureName</td>
<td>The name of the licensed feature. Possible features include: Server</td>
</tr>
<tr>
<td>productName</td>
<td>The name of the product with the licensed feature. Possible products include: ElectricCommander</td>
</tr>
</tbody>
</table>

Positional arguments

productName, featureName

Response

None or a status OK message.

ec-perl

syntax: $cmdr->deleteLicense(<productName>, <featureName>);

Example

$cmdr->deleteLicense("ElectricCommander", "Server");

ectool

syntax: ectool deleteLicense <productName> <featureName>
**Example**

```
ectool deleteLicense ElectricCommander Server
```

---

**getServerStatus**

Retrieves the current status of the ElectricCommander server.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>diagnostics</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>• threadDump - stack dumps of all threads in the server</td>
</tr>
<tr>
<td></td>
<td>• statistics - output from all system timers</td>
</tr>
<tr>
<td></td>
<td>• systemProperties - values of all java system properties</td>
</tr>
<tr>
<td></td>
<td>• environmentVariables - values of all environment variables</td>
</tr>
<tr>
<td></td>
<td>• settings - values of all server settings</td>
</tr>
<tr>
<td></td>
<td>• serverInfo - output from getServerInfo call.</td>
</tr>
<tr>
<td>serverStateOnly</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>timeout</td>
<td>This flag specifies the timeout for the element flag. The default value is 120 seconds.</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Returns the current status of the server, including the log message generated during the startup sequence.

This command returns different information depending on when and how it is called.

**Note:** You will get a lengthy response if you connect with a session that has admin privileges or if the server is still in a bootstrap state. After the server enters the "running" state, it is able to perform access checks but displays only the short form until you log in.

A simple response:

```
<serverState>running</serverState>
```
For more detailed server status response information, click here.

**ec-perl**

*Syntax:* $cmdr->getServerStatus({<optionals>});

*Examples*

```
$cmdr->getServerStatus();
$cmdr->getServerStatus({diagnostics=>1});
```

**ectool**

*Syntax:* ectool getServerStatus

*Examples*

```
ectool getServerStatus

ectool getServerStatus --diagnostics 1
```

Back to Top
API Commands - Tier Map

createTierMap
deleteTierMap
deleteTierMapping
getTierMaps
modifyTierMap

createTierMap

Creates a new tier map for an application.

Required Arguments

projectName

Description: Name for the project; must be unique among all projects.

Argument Type: String

applicationName

Description: Name of the application; must be unique among all applications in the project.

Argument Type: String

environmentProjectName

Description: Name of the environment's project; must be unique among all projects.

Argument Type: String

environmentName

Description: Name of the environment; must be unique among all applications in the project.

Argument Type: String

Optional Arguments

tierMapName

Description: The name of the tier map. If not specified, the operation will generate a name of the form as follows: <applicationName>-<environmentName>.

Argument Type: String

tierMapping

Description: List of mappings between the application tiers and the environment tiers. The list shows the mappings as <applicationTier>=<environmentTier>.

Argument Type: Map

Response

Returns a tier-map element.
**ec-perl**

Syntax:

```bash
$<object>->createTierMap(<projectName>, <applicationName>,
<environmentProjectName>, <environmentName>), {<optionals>});
```

Example:

```bash
$ec->createTierMap("default", "newApp", "defaultEnv", "Env1",
{tierMapping => [ {applicationTier => "AppTier1",
environmentTier => "EnvTier1"}, {applicationTier => "AppTier2",
environmentTier => "EnvTier2"}], tierMapName => "TierMap1"});
```

**ectool**

Syntax:

```bash
ectool createTierMap <projectName> <applicationName>
<environmentProjectName> <environmentName> [optionals...]
```

Example:

```bash
ectool createTierMap default newApp defaultEnv Env1 --tierMapName TierMap1
--tierMapping AppTier1=EnvTier1 AppTier2=EnvTier2
```

**deleteTierMap**

Deletes a tier map from an application.

**Required Arguments**

- **projectName**
  - **Description:** Name for the project; must be unique among all projects.
  - **Argument Type:** String

- **applicationName**
  - **Description:** Name of the application; must be unique among all applications in the project.
  - **Argument Type:** String

- **environmentProjectName**
  - **Description:** Name of the environment's project; must be unique among all projects.
  - **Argument Type:** String

- **environmentName**
  - **Description:** Name of the environment; must be unique among all applications in the project.
  - **Argument Type:** String

**Optional Arguments**

None

**Response**

None or a status OK message.
**ec-perl**

Syntax:

```perl
$<object>-deleteTierMap(<projectName>, <applicationName>, <environmentProjectName>, <environmentName>);
```

Example:

```perl
$ec->deleteTierMap("default", "App1", "MyProj", "Env1");
```

**ectool**

Syntax:

```bash
ectool deleteTierMap <projectName> <applicationName> <environmentProjectName> <environmentName>
```

Example:

```bash
ectool deleteTierMap default TierMapToDelete defaultEnv Env1
```

d**eleteTierMapping**

Deletes a tier mapping from a tier map.

**Required Arguments**

- `projectName`
  
  **Description:** Name for the project; must be unique among all projects.
  
  **Argument Type:** String

- `applicationName`
  
  **Description:** Name of the application; must be unique among all applications in the project.
  
  **Argument Type:** String

- `environmentProjectName`
  
  **Description:** Name of the environment's project; must be unique among all projects.
  
  **Argument Type:** String

- `environmentName`
  
  **Description:** Name of the environment; must be unique among all applications in the project.
  
  **Argument Type:** String

- `applicationTierName`
  
  **Description:** Name of the application tier.
  
  **Argument Type:** String

**Optional Arguments**

None

**Response**

Deletes the specified tier mapping.
**deleteTierMapping**

Syntax:

```perl
deleteTierMapping(<projectName>, <applicationName>, <environmentProjectName>, <environmentName>, <applicationTierName>);
```

Example:

```perl
deleteTierMap("default", "App1", "MyProj", "Env1", "InstallTier");
```

**ectool**

Syntax:

```bash
ectool deleteTierMapping <projectName> <applicationName> <environmentProjectName> <environmentName> <applicationTierName>
```

Example:

```bash
ectool deleteTierMapping default TierMapToDelete defaultEnv Env1 InstallTier
```

**getTierMaps**

Retrieves all tier maps that are used by the given application.

**Required Arguments**

**projectName**

Description: Name for the project; must be unique among all projects.

Argument Type: String

**applicationName**

Description: Name of the application; must be unique among all projects.

Argument Type: String

**Optional Arguments**

None

**Response**

Returns a list of tier maps.

**ec-perl**

Syntax:

```perl
getTierMaps(<projectName>, <applicationName>);
```

Example:

```perl
getTierMaps("default", "NewApp");
```

**ectool**

Syntax:

```bash
getTierMaps <projectName> <applicationName>
```

Example:
modifyTierMap

Modifies an existing tier map.

**Required Arguments**

- **projectName**
  - **Description:** Name for the project; must be unique among all projects.
  - **Argument Type:** String

- **applicationName**
  - **Description:** Name of the application; must be unique among all applications in the project.
  - **Argument Type:** String

- **environmentProjectName**
  - **Description:** Name of the environment's project; must be unique among all projects.
  - **Argument Type:** String

- **environmentName**
  - **Description:** Name of the environment; must be unique among all applications in the project.
  - **Argument Type:** String

**Optional Arguments**

- **tierMapName**
  - **Description:** New name of the tier map, if specified.
  - **Argument Type:** String

- **tierMapping**
  - **Description:** List of mappings between the application tiers and the environment tiers. The list shows the mappings as `<applicationTier>=<environmentTier>`.
  
  If you use this argument, new tier mappings are added or existing mappings are updated for the specified application tiers. This argument does *not* replace all the mappings and thus does *not* remove the mappings that were not specified in the API call. To remove mappings, use the `deleteTierMapping` command.
  
  - **Argument Type:** Map

**Response**

- Retrieves the updated tier map.

ec-perl

- **Syntax:**
modifyTierMap(<projectName>, <applicationName>,
<environmentProjectName>, <environmentName>), {<optionals>});

Example:
modifyTierMap("default", "newApp", "defaultEnv", "Env1",
{tierMapping => [{applicationTier => "AppTier1",
environmentTier => "EnvTier1"}, {applicationTier => "AppTier2",
environmentTier => "EnvTier2"}], tierMapName => "TierMap1"});

tool
Syntax:
tool modifyTierMap <projectName> <applicationName>
<environmentProjectName> <environmentName> [optionals...]

Example:
tool modifyTierMap default newApp defaultEnv Env1 --tierMapName TierMap1
--tierMapping AppTier1=EnvTier1 AppTier2=EnvTier2
addUsersToGroup

Adds ones or more specified users to a particular group.

You must specify a groupName and one or more user names.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName</td>
<td>The name of the group you are modifying.</td>
</tr>
<tr>
<td>usernames</td>
<td>The list of user names to add to this group.</td>
</tr>
</tbody>
</table>

Positional arguments
groupName, usernames

Response
None or status OK message.

cp-perl

**syntax:** $cmdr->addUsersToGroup(
<groupName>,

{userName=>[
<userName1>, ...
]});

**Example**

```
$cmdr->addUsersToGroup("Developers", 
{userName => "John", "Jim", "Joey"});
```

ectool

**syntax:** ectool addUsersToGroup <groupName> --userNames <userName1> ...
(Note the plural form for the userNames option)

**Example**

```
ectool addUsersToGroup Developers --userNames John Jim Joey
```
**createGroup**

Creates a new local group of users.

You must specify a `groupName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>groupName</code></td>
<td>A name you choose for the new group you are creating.</td>
</tr>
<tr>
<td><code>userNames</code></td>
<td>One or more user names to add to the group.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`groupName`

**Response**

None or status OK message.

**ec-perl**

*Syntax:*

```bash
$cmdr->createGroup(<groupName>, [<optionals>]);
```

**Example**

```bash
$cmdr->createGroup("Build Users", {userName => ['aallen', 'Betty Barker', 'cclark']});
```

**ectool**

*Syntax:*

```bash
ectool createGroup <groupName> --userNames <user1> ...
```

(Note the plural form of `userNames`.)

**Example**

```bash
ectool createGroup "Build Users" --userNames "aallen" "Betty Barker" "cclark"
```

Back to Top

**createUser**

Creates a new local user.

**Note:** This API does not apply to non-local users.

**User or Group Lists**

The commands `createUser` and `modifyUser` can have an optional argument called `groupNames`. The commands `createGroup` and `modifyGroup` can have an optional argument named `userNames`. In each case, the optional argument is followed by a list of groups or names.

Using `ectool`, your command string would be:

```bash
ectool createGroup "New Group Name" --userNames "A Adams" "B Barker"
```
To make this call via the Perl API, create a list of names and then pass a reference to the list as an optional parameter.  

**Note:** The name of the optional parameter is singular, "userName" or "userGroup," not the plural form used by ectool.

Here is an example using the Perl API:

```perl
# Run the procedure - pass a reference to the list of names
$xPath = $cmdr->createGroup("New Group Name", {
    "userName" => ['A Adams', 'B Burns'] });
```

You must specify a `userName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>The new user's email address.</td>
</tr>
<tr>
<td>fullUserName</td>
<td>The user's full name - not his or her nickname.</td>
</tr>
<tr>
<td>groupNames</td>
<td><code>&lt;group1 group2&gt;</code> Any group name containing spaces must be enclosed in double-quotes.</td>
</tr>
<tr>
<td>password</td>
<td>The new user's password.</td>
</tr>
<tr>
<td>userName</td>
<td>This could be the user's full name, but more commonly it is the shortened name, first initial and last name, or nickname used for email.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`userName`

**Response**

None or a status OK message.

**ec-perl**

`syntax:` $cmdr->createUser(<userName>, [{optionals}]);

*Example*

$cmdr->createUser("aallen", {fullUserName => "Albert Allen"});

**ectool**

`syntax:` ectool createUser <userName> ...

*Examples*

ectool createUser "aallen" --fullUserName "Albert Allen"

ectool createUser "Betty Barker"

[Back to Top]
deleteGroup

Deletes a local group.

You must specify a groupName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName</td>
<td>The name of the group you want to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

groupName

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `$cmdr->deleteGroup(<groupName>);`

*Example*

```perl
$cmdr->deleteGroup("Build Users");
```

**ectool**

*Syntax:* `ectool deleteGroup <groupName>`

*Example*

```bash
ectool deleteGroup "Build Users"
```

deleteUser

Deletes a local user.

You must specify the userName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>The name of the user you want to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

userName

**Response**

None or a status OK message.
ec-perl

**syntax:** $cmdr->deleteUser(<userName>);

**Example**

$cmdr->deleteUser("Betty Barker");

ectool

**syntax:** ectool deleteUser <userName>

**Example**

ectool deleteUser "Betty Barker"

**getGroup**

Retrieves a group by its name.

You must specify the **groupName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>groupName</strong></td>
<td>The name of the group to retrieve.</td>
</tr>
<tr>
<td><strong>providerName</strong></td>
<td>Using this option allows you to search only the specified provider for group information. (LDAP or Active Directory)</td>
</tr>
</tbody>
</table>

**Positional arguments**

**groupName**

**Response**

One **group** element.

ec-perl

**syntax:** $cmdr->getGroup(<groupName>, {<optionals>});

**Example**

$cmdr->getGroup("myGroup", {providerName => "LDAP"});

ectool

**syntax:** ectool getGroup <groupName> ...

**Example**

ectool getGroup myGroup --providerName LDAP
getGroups

Retrieves all groups.

Arguments | Descriptions
---|---
filter | A string used to filter the returned groups by their names.
includeAll | `<Boolean flag - 0|1|true|false>` When enabled, this argument returns ALL matching groups, including LDAP or non-LDAP groups that may or may not be in the Commander database already. A group is added to the Commander database when a user [who is a member of that group] logs in to Commander for the first time.
maximum | Specifies the maximum number of groups you want to see.

Positional arguments

None

Response

Zero or more `group` elements, each containing summary information only.

**ec-perl**

*Syntax:* $cmdr->getGroups({<optionals>});

*Example*

$cmdr->getGroups({filter => " dev*", maximum => 3});

**ectool**

*Syntax:* ectool getGroups ...

*Example*

ectool getGroups --filter dev* --maximum 3

Back to Top

getUser

Retrieves a user by name.

You must specify the `userName`.

Arguments | Descriptions
---|---
providerName | The name of the directory provider. If specified, this option limits the search to the specified directory provider.
userName | The name of the user.
Positional arguments

userName

Response

One user element.

cp-perl

Syntax: $cmdr->getUser(<userName>, {<optionals>});

Example

$cmdr->getUser("Betty Barker");

ectool

Syntax: ectool getUser <userName> ... 

Example

ectool getUser "Betty Barker"

getUsers

Retrieves users. By default, this command returns users who have been added to the Commander database, which means they have logged in previously.

Note: When calling getUsers, the default limit is 100 user records. Use the maximum option to specify a larger number, but this may inhibit performance, or you could define a search pattern to filter your search and conduct multiple queries.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>&lt;filter pattern&gt; Supply a filter pattern to match user names. The filter is not case sensitive and can include the &quot;*&quot; wildcard character.</td>
</tr>
<tr>
<td>includeAll</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>maximum</td>
<td>&lt;number of users&gt; Specify a larger number of user records to retrieve. The default limit is 100 user records.</td>
</tr>
</tbody>
</table>

Positional arguments

None

Response

Zero or more user elements with summary information only.
ec-perl

**syntax:** $cmdr->getUsers({<optionals>});

**Examples**

$cmdr->getUsers();

$cmdr->getUsers({filter => '*Betty*', maximum => 25});

ectool

**syntax:** ectool getUsers ...

**Examples**

ectool getUsers

ectool getUsers --filter *Betty* --maximum 25

Back to Top

**login**

Logs into the server and saves the session ID for subsequent ectool use. The user name provided determines the permissions for commands that can be run during the session.

You must specify the **userName** and **password**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>The password for the user who is &quot;logging in&quot;.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of a user who has login privileges.</td>
</tr>
</tbody>
</table>

**Positional arguments**

userName, password

**Response**

One session element containing the session ID.

ec-perl

**syntax:** $cmdr->login(<userName>,<password>);

**Example**

$cmdr->login("Ellen Ernst", "ee123");

ectool

**syntax:** ectool login <userName> <password>

**Note:** ectool will prompt for the password if not supplied.
Example
ectool --server EAVMXP login "Ellen Ernst" "ee123"

logout
Logs out of the client session.

Arguments | Descriptions
---|---
None |

Positional arguments
None

Response
None or a status OK message.

ec-perl

Example
$cmdr->logout();

ectool

Example
ectool logout

modifyGroup
Modifies an existing group.
You must specify `groupName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>groupName</code></td>
<td>The name of the group to modify.</td>
</tr>
<tr>
<td><code>migrateSettings</code></td>
<td><code>&lt;targetGroupName&gt;</code> Use this argument to specify the new name to which the settings need to be moved.</td>
</tr>
<tr>
<td><code>newName</code></td>
<td>Supply any name of your choice to rename the group.</td>
</tr>
<tr>
<td><code>removeAllUsers</code></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>
### Arguments

| userNames       | Provide a complete list of names for the group. These names will replace existing names in the group. Any name with spaces must be enclosed in double-quotes. |

#### Positional arguments

- **groupName**

#### Response

None or a status OK message.

#### ec-perl

**syntax:** $cmdr->modifyGroup(<groupName>, {...});

**Examples**

$cmdr->modifyGroup("Build Users", {userName => "dduncan"});

$cmdr->modifyGroup("Build Users", {userName => ["dduncan", "jack"]});

#### ectool

**syntax:** ectool modifyGroup <groupName> ...

**Examples**

ectool modifyGroup "Build Users" --userNames dduncan

ectool modifyGroup "Build Users" --userNames dduncan jack

### modifyUser

Modifies an existing *local* user.

**Note:** This API does *not* apply to non-local users.

#### User or Group Lists

The commands **createUser** and **modifyUser** can have an optional argument called *groupNames*. The commands **createGroup** and **modifyGroup** can have an optional argument named *userNames*. In each case, the optional argument is followed by a list of groups or names.

Using ectool, your command string would be:

ectool createGroup "New Group Name" --userNames "A Adams" "B Barker"

To make this call via the Perl API, create a list of names and then pass a reference to the list as an optional parameter.

**Note:** The name of the optional parameter is singular, "userName" or "userGroup," not the plural form used by ectool.

Here is an example using the Perl API:

```perl
# Run the procedure - pass a reference to the list of names
```
$xPath = $cmdr->createGroup("New Group Name", {  
  "userName" => ['A Adams', 'B Burns'] });

You must specify a `userName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>The user's email address.</td>
</tr>
<tr>
<td>fullUserName</td>
<td>The user's full name. For example, &quot;John Smith&quot;.</td>
</tr>
<tr>
<td>groupNames</td>
<td><code>group1 [group2 ...]</code> Assigns the user to one or more groups and removes the user from any groups not included in the list.</td>
</tr>
<tr>
<td>migrateSettings</td>
<td><code>&lt;targetUserName&gt;</code> Use this option to specify the new name to which the settings need to be moved.</td>
</tr>
<tr>
<td>newName</td>
<td>The user's new name (for example, if changing an existing user's surname).</td>
</tr>
<tr>
<td>password</td>
<td>Supply a new password to set for the user.</td>
</tr>
<tr>
<td>removeFromAllGroups</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>sessionPassword</td>
<td>If changing the user's password, you must supply the password used in the &quot;login&quot; command also.</td>
</tr>
<tr>
<td>userName</td>
<td>The name used by the user to login and/or receive email. For example, &quot;jsmith&quot;.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`userName`

**Response**

None or a status OK message.

**ec-perl**

```
syntax: $cmdr->modifyUser(<userName>, {<optionals>});

Example

$cmdr->modifyUser("Betty Barker", {email => "bbarker@abc.com"});
```

**ectool**

```
syntax: ectool modifyUser <userName> ...

Example

ectool modifyUser "Betty Barker" --email "bbarker@abc.com"
```

Back to Top
removeUsersFromGroup

Removes one or more users from a particular group.

You must specify a groupName and one or more user names.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName</td>
<td>The name of the group from which to remove users.</td>
</tr>
<tr>
<td>usernames</td>
<td>The list of users to remove from the group.</td>
</tr>
</tbody>
</table>

**Positional arguments**

groupName, usernames

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* $cmdr->removeUsersFromGroup(<groupName>, {<optionals>});

*Example*

$cmdr->removeUsersFromGroup("Developers", {userName => ["John", "Jim", "Joey"]});

**ectool**

*Syntax:* ectool removeUsersFromGroup <groupName> <userNames> ...

*Example*

ectool removeUsersFromGroup Developers --userNames John Jim Joey

Back to Top
API Commands - Workflow Management

completeWorkflow

Marks a workflow as completed. When completed, transitions are no longer evaluated.

You must specify projectName and workflowName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

Positional arguments

projectName, workflowName

Response

None or status OK message.

ec-perl

.syntax: $cmdr->completeWorkflow (<projectName>, <workflowName>{...});

Example

$cmdr->completeWorkflow ("projectA", "workflow_26_201010121647");

decTool

.syntax: ectool completeWorkflow <projectName> <workflowName>

Example

ectool completeWorkflow projectA workflow_26_201010121647

deleteWorkflow

Deletes a workflow, including all states and transitions.
You must specify a `projectName` and a `workflowName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteProcesses</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow to delete.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `projectName`, `workflowName`

**Response**

None or status OK message.

**ec-perl**

*Syntax:* $cmdr->deleteWorkflow (<projectName>, <workflowName>);

*Example*

$cmdr->deleteWorkflow ("projectA", "workflow_26_201010121647");

**ectool**

*Syntax:* ectool deleteWorkflow <projectName> <workflowName> ...

*Example*

ectool deleteWorkflow projectA workflow_26_201010121647

---

**getState**

Finds a state by name.

You must specify `projectName`, `workflowName`, and `stateName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the state.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `projectName`, `workflowName`, `stateName`
Response
One state element.

ec-perl
syntax: $cmdr->getState (<projectName>, <workflowName>, <stateName>);

Example
$cmdr->getState ("projectA", "workflow_26_201010121647", "build");

ectool
syntax: ectool getState <projectName> <workflowName> <stateName>

Example
ectool getState projectA workflow_26_201010121647 build

getStates
Retrieves all states in a workflow.
You must specify projectName and workflowName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the state.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

Positional arguments
projectName, workflowName

Response
One or more state elements.

ec-perl
syntax: $cmdr->getStates (projectName>, <workflowName>);

Example
$cmdr->getStates ("projectA", "workflow_26_201010121647");

ectool
syntax: ectool getStates <projectName> <workflowName>

Example
ectool getStates projectA workflow_26_201010121647
getTransition

Finds a transition by name.

You must specify **projectName, workflowName, stateName, and transitionName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the transition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**
- projectName, workflowName, stateName, transitionName

**Response**
- One transition element.

**ec-perl**

*Syntax:* $cmdr->getTransition (projectName>, <workflowName>, <stateName>,<transitionName>);

*Example*

$cmdr->getTransition ("projectA", "workflow_26_201010121647", "build", "build2test");

**ectool**

*Syntax:* ectool getTransition <projectName> <workflowName> <stateName> <transitionName>

*Example*

ectool getTransition projectA workflow_26_201010121647 build build2test

Back to Top

getTransitions

Retrieves all transitions in a workflow.

You must specify **projectName, workflowName, and stateName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the transition.</td>
</tr>
</tbody>
</table>

Back to Top
## API Commands - Workflow Management

### getTransitions

Finds transition definitions for a state.

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>targetState</td>
<td>The target state for the transition definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- projectName, workflowName, stateName

**Response**

One or more transition elements.

**ec-perl**

*Syntax:* $cmdr->getTransitions (<projectName>, <workflowName>, <stateName>);

*Example*

$cmdr->getTransitions ("projectA", "workflow_26_201010121647", "build");

**ectool**

*Syntax:* ectool getTransitions <projectName> <workflowName> <stateName>

*Example*

ectool getTransitions projectA workflow_26_201010121647 build

### getWorkflow

Finds a workflow by name.

You must specify a projectName and workflowName.

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- projectName, workflowName

**Response**

One workflow element.
getWorkflows

Retrieves all workflow instances in a project.
You must specify a projectName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflows.</td>
</tr>
</tbody>
</table>

Positional arguments

projectName

Response

Zero or more workflow elements.

runWorkflow

Runs the specified workflow definition and returns the workflow name.
You must specify the `projectName` and `workflowDefinitionName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the workflow starting state. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the starting state.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow definition.</td>
</tr>
<tr>
<td>startingState</td>
<td>The initial state of the workflow.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, workflowDefinitionName

**Response**

The workflow name is returned.

**ec-perl**

.syntax: $cmdr->runWorkflow (<projectName>, <workflowDefinitionName>, [<optionals>]);

**Example**

$cmdr->runWorkflow ("projectA", "BTD", {startingState => "build"});

**ectool**

.syntax: ectool runWorkflow <projectName> <workflowDefinitionName> ...

**Example**

ectool runWorkflow projectA BTD --startingState build

**transitionWorkflow**

Manually transition from the active workflow state.

You must specify `projectName`, `workflowName`, `stateName`, and `transitionName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the transition’s target state. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the target state.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow to transition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow to transition.</td>
</tr>
</tbody>
</table>

**Positional arguments**
projectName, workflowName, stateName, transitionName

**Response**
None or status OK message.

**ec-perl**

*Syntax:* $cmdr->transitionWorkflow (<projectName>, <workflowName>, <stateName>, <transitionName>, {<optionals>});

*Example*
$cmdr->transitionWorkflow ("projectA", "workflow_26_201010121647", "build", "build2 test");

**ectool**

*Syntax:* ectool transitionWorkflow <projectName> <workflowName> <stateName> <transitionName> ...

*Example*
ectool transitionWorkflow projectA workflow_26_201010121647 build build2test

Back to Top
createStateDefinition

Creates a new state definition for a workflow definition. Optionally, a state may launch either a procedure or a sub-workflow as its "process" when the state is entered.

You must specify projectName, workflowDefinitionName, and stateDefinitionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the process. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the process. For more information about parameters, click here.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project.</td>
</tr>
<tr>
<td>startable</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>Choose any unique name of your choice for the state definition. This name must be unique within the workflow definition.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>Name of the procedure launched when the state is entered. Also requires subproject</td>
</tr>
<tr>
<td>subproject</td>
<td>Name of the project containing the procedure or workflow launched when the state is entered.</td>
</tr>
</tbody>
</table>
createStateDefinition

Creates a new state definition for a workflow definition.

You must specify `projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName, and targetState.`
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the target state. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the target state. For more information about parameters, click here.</td>
</tr>
<tr>
<td>condition</td>
<td>A fixed text or text embedding property references that are evaluated into a logical TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE. This field is ignored by the server if trigger is set to manual.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt;</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>targetState</td>
<td>Target state for the transition definition.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>Choose any unique name of your choice for the transition definition. This name must be unique within the state definition.</td>
</tr>
<tr>
<td>trigger</td>
<td>Possible values are: onEnter</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**
- projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName, targetState

**Response**
- One transitionDefinition element.

**ec-perl**

**syntax:**
```
$cmdr->createTransitionDefinition ('<projectName>', '<workflowDefinitionName>',
'<stateDefinitionName>', '<transitionDefinitionName>', '<targetState>',
{<optionals>});
```

**Example**
```
$cmdr->createTransitionDefinition ('ProjectA', 'BTD', 'build', 'build2test', 'test',
{trigger => 'manual', description => 'free text'});
```
ectool

**syntax:**

```
ectool createTransitionDefinition <projectName> <workflowDefinitionName> 
<stateDefinitionName> <transitionDefinitionName> <targetState> ...
```

**Example**

```
ectool createTransitionDefinition ProjectA BTD build build2test test --trigger manual 
--description "free text"
```

### createWorkflowDefinition

Creates a new workflow definition for a project.

You must supply a `projectName` and a `workflowDefinitionName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with</td>
</tr>
<tr>
<td></td>
<td><code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;/br&gt;</code>, <code>&lt;/div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>Choose any unique name of your choice for the workflow definition. This name must be unique within the project.</td>
</tr>
<tr>
<td>workflowTemplateName</td>
<td>The name of the workflow template.</td>
</tr>
</tbody>
</table>

#### Positional arguments

`projectName, workflowDefinitionName`

#### Response

One `workflowDefinition` element.

### ec-perl

**syntax:**

```
$cmdr->createWorkflowDefinition {projectName>, <workflowDefinitionName>,
{<optionals>>};
```

**Example**

```
$cmdr->createWorkflowDefinition ("projectA", "BTD", {description => "free text"});
```

### ectool

**syntax:**

```
ectool createWorkflowDefinition <projectName> <workflowDefinitionName> ...
```
deleteStateDefinition

Deletes a state definition.

You must specify a projectName, workflowDefinitionName, and stateDefinitionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the state definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, workflowDefinitionName, stateDefinitionName

**Response**

None or status OK message.

**ec-perl**

* syntax:*

```perl
$cmdr->deleteStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>);
```

**Example**

```perl
$cmdr->deleteStateDefinition ("projectA", "BTD", "build");
```

**ectool**

* syntax:*

```bash
ectool deleteStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName>
```

**Example**

```bash
ectool deleteStateDefinition projectA BTD build
```

Back to Top

deleteTransitionDefinition

Deletes a transition definition.

You must specify a projectName, workflowDefinitionName, stateDefinitionName, and transitionDefinitionName.
deleteTransitionDefinition

Deletes a transition definition.

You must specify a **projectName** and a **workflowDefinitionName**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the transition definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

Positional arguments

**projectName**, **workflowDefinitionName**, **stateDefinitionName**, **transitionDefinitionName**

Response

None or status OK message.

**ec-perl**

**Syntax:**

```
$cmdr->deleteTransitionDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, <transitionDefinitionName>);
```

**Example**

```
$cmdr->deleteTransitionDefinition ("projectA", "BTD", "build", "build2test");
```

**ectool**

**Syntax:**

```
ectool deleteTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName>
```

**Example**

```
ectool deleteTransitionDefinition projectA BTD build build2test
```

---

deleteWorkflowDefinition

Deletes a workflow definition, including all state and transition definitions.

You must specify a **projectName** and a **workflowDefinitionName**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow definition to delete.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

Positional arguments

**projectName**, **workflowDefinitionName**

---
Response
None or status OK message.

ec-perl

_syntax:_ $cmdr->deleteWorkflowDefinition (<projectName>, <workflowDefinitionName>);

_Example_

$cmdr->deleteWorkflowDefinition ("projectA", "BTD");

ectool

_syntax:_ ectool deleteWorkflowDefinition <projectName> <workflowDefinitionName>

_Example_

ectool deleteWorkflowDefinition projectA BTD

get-StateDefinition

Finds a state definition by name.
You must specify projectName, workflowDefinitionName, and stateDefinitionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the state definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

Positional arguments

projectName, workflowDefinitionName, stateDefinitionName

Response
One stateDefinition element.

ec-perl

_syntax:_ $cmdr->getStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>);

_Example_

$cmdr->getStateDefinition ("projectA", "BTD", "build");

ectool

_syntax:_ ectool getStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName>
**Example**

`ectool getStateDefinition projectA BTD build`

---

**getStateDefinitions**

Retrieves all state definitions in a workflow definition.

You must specify **projectName** and **workflowDefinitionName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeFormalParameters</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the state definition.</td>
</tr>
<tr>
<td>startableOnly</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, workflowDefinitionName`

**Response**

One or more `stateDefinition` elements.

**ec-perl**

*Syntax:*  
$$cmdr->getStateDefinitions ({<projectName>, <workflowDefinitionName>, <optionals>});$$

*Example*

```
$cmdr->getStateDefinitions ("projectA", "BTD", {startableOnly => 1});
```

**ectool**

*Syntax:*  
```
ectool getStateDefinitions <projectName> <workflowDefinitionName> ...
```

*Example*

```
ectool getStateDefinitions projectA BTD --startableOnly 1
```

---

**getTransitionDefinition**

Finds a transition definition by name.

You must specify **projectName**, **workflowDefinitionName**, **stateDefinitionName**, **transitionDefinitionName**.
Arguments | Descriptions
---|---
projectName | The name of the project containing the transition definition.
stateDefinitionName | The name of the state definition.
transitionDefinitionName | The name of the transition definition.
workflowDefinitionName | The name of the workflow definition.

### Positional arguments
projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName

### Response
One transitionDefinition element.

**ec-perl**

**syntax:**
```
$cmdr->getTransitionDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, <transitionDefinitionName>);
```

**Example**
```
$cmdr->getTransitionDefinition ("projectA", "BTD", "build", "build2test");
```

**ectool**

**syntax:**
```
ectool getTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName>
```

**Example**
```
ectool getTransitionDefinition projectA BTD build build2test
```

### getTransitionDefinitions

Retrieves all transition definitions in a workflow definition.

You must specify projectName, stateDefinitionName, workflowDefinitionName.

Arguments | Descriptions
---|---
projectName | The name of the project containing the transition definitions.
stateDefinitionName | The name of the state definition.
targetState | The name of the target state.
workflowDefinitionName | The name of the workflow definition.
Positional arguments

projectName, stateDefinitionName, workflowDefinitionName

Response

Zero or more transitionDefinition elements.

cPerl

syntax: $cmdr->getTransitionDefinitions (<projectName>, <stateDefinitionName>, <workflowDefinitionName>, {...optionals});

Example

$cmdr->getTransitionDefinitions ("projectA", "build", "BTD");

ectool

syntax: ectool getTransitionDefinitions <projectName> <stateDefinitionName> <workflowDefinitionName> ...

Example

ectool getTransitionDefinitions projectA build BTD

getWorkflowDefinition

Finds a workflow definition by name.

You must specify a projectName and a workflowDefinitionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

Positional arguments

projectName, workflowDefinitionName

Response

One workflowDefinition element.

cPerl

syntax: $cmdr->getWorkflowDefinition (<projectName>, <workflowDefinitionName>);

Example

$cmdr->getWorkflowDefinition ("projectA", "BTD");

ectool

syntax: ectool getWorkflowDefinition <projectName> <workflowDefinitionName>
getWorkflowDefinitions

Retrieves all workflow definitions in a project.
You must specify a projectName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow definitions.</td>
</tr>
</tbody>
</table>

Positional arguments

projectName

Response

Zero or more workflowDefinition elements.

e-c-perl

syntax: $cmdr->getWorkflowDefinitions (<projectName>);

Example

$cmdr->getWorkflowDefinitions ("projectA");

ectool

syntax: ectool getWorkflowDefinitions <projectName>

Example

ectool getWorkflowDefinitions projectA

modifyStateDefinition

Modifies an existing state definition.
You must specify projectName, workflowDefinitionName, and stateDefinitionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the process. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called process.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; </code>&lt;p&gt; <code>&lt;pre&gt; </code>&lt;span&gt; <code>&lt;style&gt; </code>&lt;table&gt; <code>&lt;tc&gt; </code>&lt;td&gt; <code>&lt;th&gt; </code>&lt;tr&gt; <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the state definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition to modify.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>The name of the procedure launched when the state is entered. <strong>Also requires</strong> subproject</td>
</tr>
<tr>
<td>subproject</td>
<td>The name of the project containing the procedure or workflow launched when the state is entered.</td>
</tr>
<tr>
<td>substartingState</td>
<td>The name of the workflow starting state that is launched when the state is entered. <strong>Also requires</strong> subproject and subworkflowDefinition</td>
</tr>
<tr>
<td>subworkflowDefinition</td>
<td>The name of the workflow definition launched when the state is entered. <strong>Also requires</strong> subproject</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**  
projectName, workflowDefinitionName, stateDefinitionName

**Response**  
One stateDefinition element.

**ec-perl**  
*Syntax:* $cmdr->modifyStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>);

*Example*  
$cmdr->modifyStateDefinition ("projectA", "BTD", "build",  
{startable => 1,  
subproject => "factory",}
modifyTransitionDefinition

Modifies an existing transition definition.

You must specify projectName, workflowDefinitionName, stateDefinitionName, and transitionDefinitionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the target state. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the target state.</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>condition</td>
<td>A fixed text or text embedded property references that are evaluated into a logical &quot;true&quot; or &quot;false&quot;. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as &quot;false&quot;. Any other result string is interpreted as &quot;true&quot;. This field is ignored by the server if trigger is set to manual.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>newName</td>
<td>A new name of your choice for the transition definition--must be a unique name within the workflow.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the transition definition to modify.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetState</td>
<td>The target state for the transition definition.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition to modify.</td>
</tr>
<tr>
<td>trigger</td>
<td>Possible values are: onEnter</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

### Positional arguments

- `projectName`, `workflowDefinitionName`, `stateDefinitionName`, `transitionDefinitionName`

### Response

One `transitionDefinition` element.

### ec-perl

**syntax:**

```perl
$cmdr->modifyTransitionDefinition (<projectName>, <workflowDefinitionName>,
<stateDefinitionName>, <transitionDefinitionName>, {<optionals>});
```

**Example**

```perl
$cmdr->modifyTransitionDefinition ("projectA", "BTD", "build", "build2test",
{targetState => "deploy",
 trigger => "onCompletion",
 description => "bypass all tests")});
```

### ectool

**syntax:**

```bash
ectool modifyTransitionDefinition <projectName> <workflowDefinitionName>
<stateDefinitionName> <transitionDefinitionName> ...
```

**Example**

```bash
ectool modifyTransitionDefinition projectA BTD build build2test
--targetState deploy
--trigger onCompletion
--description "bypass all tests"
```

### Back to Top

### modifyWorkflowDefinition

Modifies an existing workflow definition.

You must specify `projectName` and `workflowDefinitionName`. 
### moveStateDefinition

Moves a state definition within a workflow definition.

You must specify `projectName`, `workflowDefinitionName`, and `stateDefinitionName`. 
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>beforeStateDefinition</td>
<td>Use this option to reorder state definitions in a workflow definition. The state definition (stateDefinitionName) will be moved to a position just before the state definition “named” by this option. If omitted, the state definition is moved to the end of the workflow definition.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the state definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition to move.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

### Positional arguments

projectName, workflowDefinitionName, stateDefinitionName

### Response

None or status OK message.

#### ec-perl

**syntax:**

```
$cmdr->moveStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, {<optionals>});
```

**Example**

```
$cmdr->moveStateDefinition ("projectA", "BTD", "deploy", 
  {beforeStateDefinition => "test"});
```

#### ectool

**syntax:**

```
ectool moveStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> ...
```

**Example**

```
ectool moveStateDefinition projectA BTD deploy --beforeStateDefinition test
```

### moveTransitionDefinition

Moves a transition definition within a workflow definition.

You must specify projectName, workflowDefinitionName, stateDefinitionName, and transitionDefinitionName.
### API Commands - Workflow Definition Management

#### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>beforeTransitionDefinition</td>
<td>Use this option to move a transition definition in a workflow definition. The transition definition is moved to a position just before the transition definition named by this option. If omitted, the transition definition is moved to the end of the workflow definition.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the transition definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition to move.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

#### Positional arguments

projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName

#### Response

None or status OK message.

**ec-perl**

**syntax:**

```perl
$cmdr->moveTransitionDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, <transitionDefinitionName>, {<optionals>});
```

**Example**

```perl
$cmdr->moveTransitionDefinition ("projectA", "BTD", "Build", "in", {beforeTransitionDefinition => "out"});
```

**ectool**

**syntax:**

```bash
ectool moveTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName> ...
```

**Example**

ectool moveTransitionDefinition projectA BTD Build in--beforeTransitionDefinition out

[Back to Top]
API Commands - Workspace Management

**createWorkspace**

Creates a new workspace.

A workspace definition consists of three paths to access the workspace in various ways:

- **agentDrivePath**
- **agentUncPath** - The agent uses `agentUncPath` and `agentDrivePath` to compute the drive mapping needed to make `agentDrivePath` valid in the step (see examples below).
- **agentUnixPath**

### Examples for `agentDrivePath` and `agentUncPath`:

<table>
<thead>
<tr>
<th>agentDrivePath</th>
<th>agentUncPath</th>
<th>Result from running a step in &quot;job123&quot; that uses this workspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:\</td>
<td>\server\share</td>
<td>The agent maps \server\share to drive n: and runs the step in n:\job123.</td>
</tr>
<tr>
<td>N:\sub1</td>
<td>\server\share\dir1\sub1</td>
<td>The agent maps \server\share\dir1 to drive n: and runs the step in n:\sub1\job123.</td>
</tr>
<tr>
<td>N:\sub1</td>
<td>\server\share\dir1</td>
<td>Invalid! No mapping can be deduced from this pair of values.</td>
</tr>
<tr>
<td>C:\ws</td>
<td>C:\ws</td>
<td>A local workspace on the agent. No drive mapping is needed. The job step runs in c:\ws\job123.</td>
</tr>
<tr>
<td>C:\ws</td>
<td></td>
<td>Same as if <code>agentUncPath</code> were set identical to <code>agentDrivePath</code>.</td>
</tr>
</tbody>
</table>

You must specify a `workspaceName`.

### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>agentDrivePath</code></td>
<td>Drive-letter-based path used by Windows agents to access the workspace in steps.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agentUncPath</td>
<td>UNC path used by Windows Commander Web servers to access the workspace. The agent uses agentUncPath and agentDrivePath to compute the drive mapping needed for making agentDrivePath valid in the step.</td>
</tr>
<tr>
<td>agentUnixPath</td>
<td>UNIX path used by UNIX agents and Linux Commander Web servers to access the workspace.</td>
</tr>
<tr>
<td>credentialName</td>
<td>Credential to use when connecting to a network location. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tt&gt; &lt;td&gt; &lt;th&gt;</td>
</tr>
<tr>
<td>local</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workspaceDisabled</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workspaceName</td>
<td>Any name you choose to name this workspace.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone where this workspace resides.</td>
</tr>
</tbody>
</table>

**Positional arguments**

workspaceName

**Response**

None or a status OK message.

**ec-perl**

**syntax:** $cmdr->createWorkspace(<workspaceName>, {<optionals>});

**Example**

$cmdr->createWorkspace('test', {agentDrivePath => 'c:/workspace', agentUncPath => 'c:/workspace', agentUnixPath => '/mnt/server/workspace'});
**Ectool**

**Syntax:**

```
ectool createWorkspace <workspaceName> ...
```

**Example**

```
 ectool createWorkspace test --agentDrivePath c:/workspace --agentUncPath c:/workspace --agentUnixPath '/mnt/server/workspace'
```

**DeleteWorkspace**

Deletes a workspace.

You must specify the `workspaceName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>workspaceName</code></td>
<td>The name of the workspace to delete.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`workspaceName`

**Response**

None or a status OK message.

**Ecp-perl**

**Syntax:**

```
$cmdr->deleteWorkspace(<workspaceName>);
```

**Example**

```
$cmdr->deleteWorkspace("test");
```

**Ectool**

**Syntax:**

```
ectool deleteWorkspace <workspaceName>
```

**Example**

```
 ectool deleteWorkspace test
```

**GetWorkspace**

Retrieves a workspace by name.

You must specify the `workspaceName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>workspaceName</code></td>
<td>The name of the workspace to retrieve.</td>
</tr>
</tbody>
</table>

---

362
### Positional arguments

workspaceName

### Response

One `workspace` element.

**ec-perl**

syntax: `$cmdr->getWorkspace(<workspaceName>);`

*Example*

```perl
$cmdr->getWorkspace("test");
```

**ectool**

syntax: `ectool getWorkspace <workspaceName>`

*Example*

```bash
ectool getWorkspace test
```

### getWorkspaces

Retrieves all workspaces.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

### Positional arguments

None

### Response

Zero or more `workspace` elements.

**ec-perl**

syntax: `$cmdr->getWorkspaces();`

*Example*

```perl
$cmdr->getWorkspaces();
```

**ectool**

syntax: `ectool getWorkspaces`

*Example*

```bash
ectool getWorkspaces
```

Back to Top
modifyWorkspace

Modifies an existing workspace.

A workspace definition consists of three paths to access the workspace in various ways:

- **agentDrivePath**
- **agentUncPath** - The agent uses agentUncPath and agentDrivePath to compute the drive mapping needed to make agentDrivePath valid in the step (see examples below).
- **agentUnixPath**

**Examples for agentDrivePath and agentUncPath:**

<table>
<thead>
<tr>
<th>agentDrivePath</th>
<th>agentUncPath</th>
<th>Result from running a step in &quot;job123&quot; that uses this workspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:\</td>
<td>\server\share</td>
<td>The agent maps \server\share to drive n: and runs the step in n:\job123.</td>
</tr>
<tr>
<td>N:\sub1</td>
<td>\server\share-dir1\sub1</td>
<td>The agent maps \server\share-dir1\sub1 to drive n: and runs the step in n:\job123.</td>
</tr>
<tr>
<td>N:\sub1</td>
<td>\server\share-dir1</td>
<td>Invalid! No mapping can be deduced from this pair of values.</td>
</tr>
<tr>
<td>C:\ws</td>
<td>C:\ws</td>
<td>A local workspace on the agent. No drive mapping is needed. The job step runs in c:\ws\job123.</td>
</tr>
<tr>
<td>C:\ws</td>
<td></td>
<td>Same as if agentUncPath were set identical to agentDrivePath.</td>
</tr>
</tbody>
</table>

You must specify a workspaceName.

**Arguments**  

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>agentDrivePath</td>
<td>Drive-letter-based path used by Windows agents to access the workspace in steps.</td>
</tr>
<tr>
<td>agentUncPath</td>
<td>UNC path used by Windows Commander web servers to access the workspace. The agent uses agentUncPath and agentDrivePath to compute the drive mapping needed for making agentDrivePath valid in the step.</td>
</tr>
<tr>
<td>agentUnixPath</td>
<td>UNIX path used by UNIX agents and Linux Commander web servers to access the workspace.</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---

credentialName | credentialName can be one of two forms: relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object. absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.
description | A plain text or HTML description for this object. If using HTML, you must surround your text with <html> ... </html> tags. The only HTML tags allowed in the text are: <a> <b> <br> <div> <dl> <i> <li> <ol> <p> <pre> <span> <style> <table> <tt> <tr> <ul>
local | <Boolean flag - 0|1|true|false> Set to "true", the workspace is local.
newName | Supply any name of your choice to rename the workspace.
workspaceName | The name of the workspace to modify.
workspaceDisabled | <Boolean flag - 0|1|true|false> Set to "true", the workspace is disabled.
zoneName | The name of the zone where this workspace resides.

Positional arguments
workspaceName

Response
None or a status OK message.

ec-perl
syntax: $cmdr->modifyWorkspace(<workspaceName>, {<optionals>});

Example
$cmdr->modifyWorkspace("test", {description => "my test workspace");

ectool
syntax: ectool modifyWorkspace <workspaceName> ...

Example
ectool modifyWorkspace test --description "my test workspace"

Back to Top
API Commands - Miscellaneous Management

**changeOwner**

Changes the owner of an object.

You must specify an object name.

**Note:** The modify privilege on the "admin" system ACL is required to change an object's owner.

For email notifiers, the owner can be changed if the current user has sufficient privileges to have deleted the object and recreated it.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms:</td>
</tr>
<tr>
<td></td>
<td>relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the</td>
</tr>
<tr>
<td></td>
<td>project that contains the request target object. Requires a qualifying</td>
</tr>
<tr>
<td></td>
<td>project name.</td>
</tr>
<tr>
<td></td>
<td>absolute (for example, &quot;projects/BuildProject/credentials/cred1&quot;) - the</td>
</tr>
<tr>
<td></td>
<td>credential can be from any specified project, regardless of the target</td>
</tr>
<tr>
<td></td>
<td>object's project.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the email configuration.</td>
</tr>
<tr>
<td>groupName</td>
<td>The full name of a group. For Active Directory and LDAP, this is a full DN.</td>
</tr>
<tr>
<td>newOwnerName</td>
<td>The name of the new owner for this object. Defaults to the current user.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin - the plugin key for a promoted plugin or a plugin</td>
</tr>
<tr>
<td></td>
<td>key and version for an unpromoted plugin.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure - may be a path to the procedure. Also requires</td>
</tr>
<tr>
<td></td>
<td>projectName</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
projectName | The name of the project - may be a path. The project name is ignored for credentials, procedure, steps, and schedules if it is specified as a path.
propertySheetId | The unique identifier for a property sheet, assigned automatically when the property sheet is created.
resourceName | The name of the resource.
scheduleName | The name of the schedule - may be a path to the schedule. Also requires projectName
stateDefinitionName | The name of the state definition.
stepName | The name of the step - may be a path to the step. Also requires projectName and procedureName
transitionDefinitionName | The name of the transition definition.
userName | The full name of the user. For Active Directory and LDAP, the name may be user@domain.
workflowDefinitionName | The name of the workflow definition.
workspaceName | The name of the workspace.

**Positional arguments**
None

**Response**
Returns the modified object.

**ec-perl**

*Syntax:* $cmdr->changeOwner({...});

*Example*

$cmdr->changeOwner ("projectName"=>"Sample Project");

**ectool**

*Syntax:* ectool changeOwner ...

*Example*

ectool changeOwner --projectName "Sample Project"

[Back to Top]
**clone**

Makes a copy of an existing ElectricCommander-platform object. For example: credential, directory provider, email configuration, email notifier, project, procedure, property sheet, resource, resource pool, schedule, state definition, step, transition definition, workflow definition, and workspace.

**Note:** You cannot clone parameters.

**IMPORTANT:**
To find the entity you want to clone, you must specify the following arguments:
- A new name for the cloned object (cloneName)
- Locator arguments

For example, if you want to clone a project, you must specify the name of the project that you want to clone.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Naming</strong></td>
<td></td>
</tr>
<tr>
<td>cloneName</td>
<td>The <code>cloneName</code> specifies the path to the new object, possibly in an alternate location. If no container path is specified, the new object is created inside the same container as the original. If no name is specified, the server will generate a name.</td>
</tr>
<tr>
<td><strong>Locators</strong></td>
<td></td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifactVersion container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the <code>emailConfig</code> container that owns the property.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential container of the property sheet which owns the property. credentialName can be one of two forms:</td>
</tr>
<tr>
<td></td>
<td><strong>relative</strong> (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td><strong>absolute</strong> (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the</td>
</tr>
<tr>
<td></td>
<td>target object's project.</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment container of the property sheet which owns the property; must be unique among all projects.</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>The name of the environment tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway container of the property sheet.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job</td>
</tr>
<tr>
<td></td>
<td>name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>The object id as returned by FindObjects.</td>
</tr>
<tr>
<td>path</td>
<td>The property path that specifies the object.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure you want to clone. <strong>Also requires</strong> projectName</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project you want to clone.</td>
</tr>
</tbody>
</table>
# Arguments and Descriptions

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertySheetId</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>providerName</td>
<td>The LDAP or Active Directory provider name.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource you want to clone.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule you want to clone.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step you want to clone.</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>System object names include:</td>
</tr>
<tr>
<td></td>
<td>admin</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user where you may need to expand the string.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace you want to clone.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>

## Positional arguments

None.

## Response

Returns the name of the new cloned object.

Using the clone command successfully depends on the context of the locator arguments in your system. The command works when the arguments are specified correctly.
**ec-perl**

**syntax:** $cmdr->clone ...;

**Examples**

# Create a copy of a procedure, as though you clicked the "Copy" button in the UI.

```perl
$xPath = $cmdr->clone(
    { projectName => "EC-Examples",
      procedureName => "set Property"
    }
);
```

# Create a copy of a procedure providing a name for the copy.

```perl
$xPath = $cmdr->clone(
    { projectName => "EC-Examples",
      procedureName => "set Property",
      cloneName => "set Property 2"
    }
);
```

# Create a copy of a procedure step.

```perl
$xPath = $cmdr->clone(
    { projectName => "EC-Examples",
      procedureName => "set Property",
      cloneName => "set Property 2",
      stepName => 'setProperty'
    }
);
```

# Copy a step using the path.

```perl
$xPath = $cmdr->clone(
    { path =>
      '/projects/EC-Examples/procedures/set Property/steps/setProperty'
    }
);
```

**ectool**

**syntax:** ectool clone ...
Examples

# Create a copy of a procedure, as though you clicked the "Copy" button in the UI.

$ ectool clone --projectName 'EC-Examples' --procedureName 'set Property'
<response requestId="1" nodeId="192.168.16.238">
  <cloneName>Set Property copy</cloneName>
</response>

# Create a copy of a procedure providing a name for the copy.

$ ectool clone --projectName 'EC-Examples' --procedureName 'set Property'
  --cloneName 'set Property 2'
<response requestId="1" nodeId="192.168.16.238">
  <cloneName>set Property 2</cloneName>
</response>

# Create a copy of a procedure step.

$ ectool clone --projectName 'EC-Examples' --procedureName 'set Property'
  --stepName 'setProperty'
<response requestId="1" nodeId="192.168.16.238">
  <cloneName>setProperty copy</cloneName>
</response>

# Create a copy of a procedure step using the full path.

$ ectool clone --path '/projects/EC-Examples/procedures/set Property/steps/setProperty'
<response requestId="1" nodeId="192.168.16.238">
  <cloneName>setProperty copy</cloneName>
</response>

Back to Top

countObjects

This API returns the count of objects specified by the provided filter.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>A list of zero or more filter criteria definitions used to define objects to find. Each element of the filter list is a hash reference containing one filter criterion. You can specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</td>
</tr>
</tbody>
</table>

Two types of filters:
* "property filters" - used to select objects based on the value of the object's intrinsic or custom property
* "boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic.

Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property.

**Property filter operators are:**
- between (2 operands)
- contains (1)
- equals (1)
- greaterOrEqual (1)
- greaterThan (1)
- in (1)
- lessOrEqual (1)
- lessThan (1)
- like (1)
- notEqual (1)
- notLike (1)
- isNotNull (0)
- isNull (0)

A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter.

**Boolean operators are:**
- not (1 operand)
- and (2 or more operands)
- or (2 or more operands)
Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectType</td>
<td>This argument specifies the type of object to count. Values include:</td>
</tr>
<tr>
<td>artifact</td>
<td>project</td>
</tr>
<tr>
<td>artifactVersion</td>
<td>property</td>
</tr>
<tr>
<td>credential</td>
<td>repository</td>
</tr>
<tr>
<td>directoryProvider</td>
<td>resource</td>
</tr>
<tr>
<td>emailconfig</td>
<td>resourcePool</td>
</tr>
<tr>
<td>emailNotifier</td>
<td>schedule</td>
</tr>
<tr>
<td>formalParameter</td>
<td>state</td>
</tr>
<tr>
<td>job</td>
<td>stateDefinition</td>
</tr>
<tr>
<td>jobStep</td>
<td>transition</td>
</tr>
<tr>
<td>logEntry</td>
<td>transitionDefinition</td>
</tr>
<tr>
<td>plugin</td>
<td>workflow</td>
</tr>
<tr>
<td>procedure</td>
<td>workflowDefinition</td>
</tr>
<tr>
<td>procedureStep</td>
<td>workspace</td>
</tr>
</tbody>
</table>

Positional arguments

objectType

Response

Returns the number of filtered objects.

ec-perl

**syntax:**

```perl
$cmdr->countObjects(<objectType>, {<optionals>});
```

**Example**

```perl
use ElectricCommander();
my @artifactNameFilters;
# Create the filter list for filtering on artifact name
push (@artifactNameFilters,
   { "propertyName"=>"artifactName",
     "operator"=>"contains",
     "operand1"=>"groupId:installer-windows",
   });
my $cmdr = new ElectricCommander();
# Perform the countObjects query
my $reference=$cmdr->countObjects("artifactVersion",
   { filter=>
     { operator=>"and",
       filter=>[
         { "propertyName"=>"modifyTime",
           operator=>"greaterOrEqual",# Give me all dates after or equal
           "operand1"=>"2014-03-25T14:48:55.286Z",
         },
         { operator => 'or', # apply 'or' for the filters in the list
           filter => \@artifactNameFilter
```
my $jobs=$reference->find('//response/count');
print $jobs;

**ectool**

Not supported.

**Back to Top**

**deleteObjects**

This API deletes objects specified by the provided filters.

Because of the complexity of specifying filter criteria, this API is not supported by ectool. However, all of its capabilities are supported through the Perl API.

You must specify an `objectType` and at least one filter.

**Note:** Currently, this API supports deleting `artifact`, `artifactVersion`, `job`, `logEntry`, `project`, `repository`, and `workflow`. 
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| filter    | Specify filters in a space-separated list: filter1 filter2 ... A list of zero or more filter criteria definitions used to define objects to find. Each element of the filter list is a hash reference containing one filter criterion. You may specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API. Two types of filters: "property filters" - used to select objects based on the value of the object's intrinsic or custom property "boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic. Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property. Property filter operators are:  
  - between (2 operands)  
  - contains (1)  
  - equals (1)  
  - greaterOrEqual (1)  
  - greaterThan (1)  
  - in (1)  
  - lessOrEqual (1)  
  - lessThan (1)  
  - like (1)  
  - notEqual (1)  
  - notLike (1)  
  - isNotNull (0)  
  - isNull (0)  
A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter. Boolean operators are:  
  - not (1 operand)  
  - and (2 or more operands)  
  - or (2 or more operands) |
| maxIds    | <id count> The maximum number of objects that will be deleted. Defaults to all objects that match the filter. |
## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>objectType</strong></td>
<td>This argument specifies the type of object to find. Values include: artifact</td>
</tr>
<tr>
<td><strong>sorts</strong></td>
<td>Specify &quot;sorts&quot; in a space-separated list: sort1 sort2 ... An ordered list of sort criteria. Each list entry consists of a property name and a sort order--either an ascending or descending sort order. If you specify more than one sort criterion, the sorts are applied according to the order they appear in the list. The first item in the list is the primary sort key. Each item in the list is a hash reference. See the code example below for instructions on forming the list and passing it to the Commander Perl API. The sort order affects which objects are deleted if a maxIds value limits the number of objects returned by the filter.</td>
</tr>
</tbody>
</table>

## Positional arguments

**objectType**

## Response

Returns a list of object references.

### ec-perl

**syntax:** $cmdr->deleteObjects(<objectType>, {<optionals>});

**Example**

This code example illustrates using a Boolean filter for the deleteObjects command to find jobs matching either of two patterns for the job name.

```perl
my @filterList;
push (@filterList, {
    "propertyName" => "jobName",
    "operator" => "like",
    "operand1" => "-%branch-%"});
push (@filterList, {
    "propertyName" => "jobName",
    "operator" => "like",
    "operand1" => "branch-%"});
my $result = $cmdr->deleteObjects('job',
    {filter => [
        { operator => 'or',
            filter => @filterList,
        }
    ]});
print "result = " . $result->findnodes_as_string("n"). "\n";
```

### ectool

Not supported.
Exports part or all server data to an XML file. By default, all data in the system is exported, although the "path" option can be used to limit the output to a single tree of objects.

If a relative filename is specified, the file is created relative to the Commander server's data directory, which by default is located:

For Windows: C:\Documents and Settings\All Users\Application Data\Electric Cloud\ElectricCommander
For Linux: /opt/electriccloud/electriccommander

You must specify a **fileName**.

**Note:** A full export/import preserves job IDs, but a partial import preserves names only, not IDs.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>compress</strong></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><strong>excludeJobs</strong></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><strong>fileName</strong></td>
<td><code>&lt;remoteFileName&gt;</code> The specified directory for the file must already exist in the system. If the path is local, it will be created on the server. If it is a network path, it must be accessible by the server and the server user.</td>
</tr>
<tr>
<td><strong>path</strong></td>
<td><code>&lt;property path&gt;</code> Specifies the path for an object to be exported. Any single object can be exported if it is specified using property path syntax. The object and its sub-objects are exported.</td>
</tr>
<tr>
<td><strong>relocatable</strong></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>safeMode</td>
<td>The <code>safeMode</code> argument determines whether the server will be quiesced before a full export begins and if yes, whether or not the server will shutdown and restarted after the export completes. Values are:</td>
</tr>
<tr>
<td></td>
<td>- none (default) - Do not quiesce the server during export.</td>
</tr>
<tr>
<td></td>
<td>- shutdown - Quiesce the server and shutdown when complete.</td>
</tr>
<tr>
<td></td>
<td>- restart - Quiesce the server and restart when complete.</td>
</tr>
<tr>
<td>Note:</td>
<td>The <code>safeMode</code> argument has no effect on partial exports.</td>
</tr>
<tr>
<td>withAcls</td>
<td>Modifies relocatable.</td>
</tr>
<tr>
<td></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>withNotifiers</td>
<td>Modifies relocatable.</td>
</tr>
<tr>
<td></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

*fileName*

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `$cmdr->export(<fileName>, {<optionals>});`

**Examples**

```perl
$cmdr->export("c:\CommanderBackup\Mar 15 2007.xml");

$cmdr->export("c:\CommanderBackup\Test Proj.xml",
             {path => "/projects[Test Proj]",
              relocatable => "true",
              withNotifiers => "true"});
```

**ectool**

*Syntax:* `ectool export <fileName> ...`

**Examples**

```bash
ectool export "c:\CommanderBackup\Mar 15 2007.xml"

ectool export "c:\CommanderBackup\Test Proj.xml" --path "/projects[Test Proj]"
   --relocatable true --withNotifiers true
```

Back to Top
findObjects

This command returns a sorted list of Commander objects based on an object type and a set of filter criteria. This API can be used to find many, but not all, types of Commander objects and is used by the Commander web interface to implement the Commander "Search" feature.

Because of the complexity of specifying filter criteria, this API is not supported by ectool. However, all of its capabilities are supported through the Perl API.

You must specify an objectType.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>A list of zero or more filter criteria definitions used to define objects to find. Each element of the filter list is a hash reference containing one filter criterion. You can specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</td>
</tr>
</tbody>
</table>
|           | Two types of filters:  
|           | "property filters" - used to select objects based on the value of the object's intrinsic or custom property |
|           | "boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic. |
|           | Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by Commander or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property. |
|           | Property filter operators are:  
|           | between (2 operands)  
|           | contains (1)  
|           | equals (1)  
|           | greaterOrEqual (1)  
|           | greaterThan (1)  
|           | in (1)  
|           | lessOrEqual (1)  
|           | lessThan (1)  
|           | like (1)  
|           | notEqual (1)  
|           | notLike (1)  
|           | isNotNull (0)  
|           | isNull (0) |
| maxIds    | A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter. |
|           | Boolean operators are:  
|           | not (1 operand)  
|           | and (2 or more operands)  
<p>|           | or (2 or more operands) |
| &lt;id count&gt; | The maximum number of object IDs that will be returned. If omitted, default behavior returns IDs for the first 1000 objects matching the query. If &quot;0&quot; is specified, the ID of every matching object is returned. |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>numObjects</td>
<td>&lt;full object count&gt; Specifies the number of full objects (not just the IDs) returned from the findObjects request. This option allows selecting a limited number of full objects to be returned in the initial request. The returned &quot;full objects&quot; correspond to the objects from the beginning of the list of object IDs. If numObjects is not specified, all full objects in the list of object IDs are returned. Any and all objects can be retrieved using the getObjects command.</td>
</tr>
<tr>
<td>objectType</td>
<td>This argument specifies the type of object to find. <strong>Values include:</strong></td>
</tr>
<tr>
<td></td>
<td>artifact</td>
</tr>
<tr>
<td>select</td>
<td>This is an unordered list of property names that specify additional top-level properties to return for each object. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</td>
</tr>
<tr>
<td>sort</td>
<td>This is an ordered list of sort criteria. Each list entry consists of a property name and a sort order—either an ascending or descending sort order. If you specify more than one sort criterion, the sorts are applied according to the order they appear in the list. The first item in the list is the primary sort key. Each item in the list is a hash reference. See the code example below for instructions on forming the list and passing it to the ElectricCommander Perl API.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**objectType**

**Response**

This command returns a list of object references. These references can be used in a subsequent call to the getObjects command. Optionally, the command can return full objects from the result list also.

**ec-perl**

**syntax:**

```
$cmdr->findObjects(<objectType>, (<optionals>));
```
Example 1

This example illustrates using a Boolean filter for the findObjects command to find jobs matching either of two patterns for the job name.

```perl
my @filterList;
push (@filterList, {"propertyName" => "jobName", "operator" => "like", "operand1" => "-%branch-%"});
push (@filterList, {"propertyName" => "jobName", "operator" => "like", "operand1" => "branch-%"});
my $result = $cmdr->findObjects('job', {filter => (@filterList,)});
print "result = ". $result->findnodes_as_string("/"). "\n";
```

Example 2

This example uses both findObjects and getObjects to manage large result sets, and also uses "select" to return the values of two properties in the returned objects.

```perl
# Search for the first 10 matching objects and retrieve the first 2
my $xPath = $cmdr->findObjects("schedule", {maxIds => "10", numObjects => "2", filter => [{propertyName => "createTime", operator => "greaterOrEqual", operand1 => "2007-01-20T00:00:00.000Z"}, {propertyName => "lastModifiedBy", operator => "like", operand1 => "adm%"}], sort => [{propertyName => "projectName", order => "ascending"}, {propertyName => "createTime", order => "descending"}], select => [{propertyName => 'prop1'}, {propertyName => 'prop2'}]});
print "Return data from Commander:\n" . $xPath->findnodes_as_string("/"). "\n";
# Build a list of all the object id's
my @allObjectsList;
my $nodeset = $xPath->find('//response/objectId');
foreach my $node (@nodeset->get_nodelist)
{
    my $objectId = $node->string_value();
push (@allObjectsList, $objectId);
}
# Retrieve the second 2 objects
my @objectList = @allObjectsList[2..3];
$xPath = $cmdr->getObjects({objectId => \@objectList});
print "Return data from Commander:\n" . $xPath->findnodes_as_string("/"). "\n";
```
Example 3
This code example illustrates composing filters combining 'or' and 'and' for finding artifacts matching either of two patterns for the artifact name, and a modify time before a specified date.

# Create the filter list for filtering on artifact name.
my &artifactNameFilters;
push &artifactNameFilters,
    {
        "propertyName" => "artifactName",
        "operator" => "equals",
        "operand1" => "groupId:installer-windows",
    },
    {
        "propertyName" => "artifactName",
        "operator" => "equals",
        "operand1" => "groupId:installer-linux"
    };

# Perform the findObjects query
my $result = $cmdr->findObjects('artifactVersion',
    {filter =>
        {operator => "and", # 'and' the different filters below
            filter => [
                #filter 1
                {  
                    "modifyTime",
                    operator => "lessOrEqual", # Give me all dates before
                    operand1 => "2011-11-01T00:00:00.000Z" # Arbitrary date
                },
                #filter 2
                {  
                    "or", # apply 'or' for the filters in the list
                    filter => &artifactNameFilters
                }
            ],
        }},
    );

print "result = ". $result->findnodes_as_string("/") . "\n";
# Top-level filters are implicitly 'and'ed, so the above findObjects query
# could also be written like this:
$result = $cmdr->findObjects('artifactVersion',
    {filter => [
        #filter 1
        {  
            "modifyTime",
            operator => "lessOrEqual", # Give me all dates before
            operand1 => "2011-11-01T00:00:00.000Z" # Arbitrary date
        },
        #filter 2
        {  
            "or", # apply 'or' for the filters in the list
            filter => &artifactNameFilters
        }
    ]};
Example 4

This example illustrates looking for a project whose name contains 'foo' and whose description equals 'bar'.

```bash
$commander->findObjects('project', {
    filter => (operator => 'and',
        filter => [{propertyName => 'projectName',
                     operator => 'contains',
                     operand1 => 'foo'},
                  {propertyName => 'description',
                     operator => 'equals',
                     operand1 => 'bar'}]);
```

Example 5

This example illustrates looking for a procedure whose project name is 'foo' and whose procedure name is either 'bar' or not 'bat'. (The top level filters are implicitly combined with 'and'.)

```bash
$commander->findObjects('procedure', {
    filter => [{propertyName => 'projectName',
                operator => 'equals',
                operand1 => 'foo'},
              {operator => 'or',
               filter => [[propertyName => 'procedureName',
                            operator => 'equals',
                            operand1 => 'bar'],
                        {operator => 'not',
                         filter => [{propertyName => 'procedureName',
                                     operator => 'equals',
                                     operand1 => 'bat'}]}]}]);
```

Example 6

This example illustrates looking for a project with certain property values.

```bash
$commander->findObjects("project", {
    filter => {operator => 'or',
           filter => [{propertyName => 'prop1',
                       operator => 'equals',
                       operand1 => 'value1'},
                    {propertyName => 'prop2',
                       operator => 'equals',
                       operand1 => 'value2'},
                    {propertyName => 'prop3',
                       operator => 'isNull'}])
```

ectool

Not supported.

Back to Top

getObjects

The `getObjects` command retrieves a list of full objects based on object IDs returned by `findJobSteps` or `findObjects`. All requested objects must be of the same objectType. See `findObjects` for a list of object types.

You must specify `objectId`.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>A list of one or more object IDs that were returned by a prior call to findObjects. Each list element is a string containing the ID. See the code example below for instructions on forming the list and passing it to the Commander Perl API.</td>
</tr>
<tr>
<td>select</td>
<td>This is an unordered list of projection definitions. Each list entry consists of a property name identifying a top-level custom property to return in addition to the rest of the object elements. See the code example below for instructions on forming the list and passing it to the Commander Perl API.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`objectId`

**Response**

A list of full objects for the requested type.

**ec-perl**

*Syntax:* $cmdr->getObjects({<optionals>});

**Example 1**

Code example for findObjects and getObjects:

```perl
use strict;
use ElectricCommander;
my $cmdr = ElectricCommander->new();

# This example runs within a Commander step, so a "login" is not needed.
use strict;
use ElectricCommander;
my $cmdr = ElectricCommander->new();

# Search for the first 10 matching objects and retrieve the first 2
my $XPath = $cmdr->findObjects("schedule",
    {maxIds => "10",
     numObjects => "2",
     filter => [{ propertyName => "createTime",
                  operator => "greaterOrEqual",
                  operand1 => "2010-01-20T00:00:00.00Z"},
                { propertyName => "lastModifiedBy",
                  operator => "like",
                  operand1 => "adm%"}],
     sort => [{ propertyName => "projectName",
                order => "ascending"},
              { propertyName => "createTime",
                order => "descending"}],
     select => [{ propertyName => 'prop1'},
                 { propertyName => 'prop2'}]});
print "Return data from Commander:\n" . $XPath->findnodes_as_string("/"). "\n";
# Build a list of all the object id's
my @allObjectsList;
my $nodeset = $XPath->find('//response/objectId');
foreach my $node ($nodeset->get_nodelist) {
```
Example 2
Code example using a Boolean filter:

```perl
my $xpath = $cmdr->findObjects('project', {
    filter => [operator => 'and',
               filter => [{propertyName => 'projectName',
                            operator => 'contains',
                            operand1 => $projectBase},
                          {propertyName => 'description',
                            operator => 'equals',
                            operand1 => 'foo'}]],
    filter => [{propertyName => 'projectName',
                operator => 'contains',
                operand1 => $projectBase}],
    filter => [{propertyName => 'description',
                operator => 'equals',
                operand1 => 'foo'}]);
```

**import**
Imports data from an XML export file.

You must specify either file or fileName.

**Note:** A full export/import preserves job IDs, but a partial import preserves names only, not IDs.
Use the preserveId option for a partial import if you need to retain the same (existing) job or workflow ID number.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>batchSize</strong></td>
<td><code>&lt;batch size&gt;</code> The number of objects imported before committing a transaction in the database. This argument limits the object batch size during import. Default value is 50 objects. If your objects are unusually large, you can throttle this number down to 1, depending on your available memory. <strong>Note:</strong> The batchSize argument is applicable to full import operations only.</td>
</tr>
<tr>
<td><strong>disableSchedules</strong></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>file</strong></td>
<td><code>&lt;localFileName&gt;</code> This is the path to a file on the client to import. The file is uploaded from the client to the server. The specified <code>&lt;file&gt;</code> value is sent as an attachment to the import API request. The server detects the presence of the attachment and reads the attached file instead of looking for a file on the server. The maximum file size specified by <code>file</code> is determined by the maximum upload-size server setting. By default the limit is 50MB, so this option should be used only for individually exported objects, not a full system export.</td>
</tr>
<tr>
<td><strong>fileName</strong></td>
<td><code>&lt;remoteFileName&gt;</code> This is the name of a file on the server to import. The file path name must be accessible to the server process on the server host.</td>
</tr>
<tr>
<td><strong>force</strong></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><strong>path</strong></td>
<td><code>&lt;property path&gt;</code> Use this argument to import a single object to a new location. For example, if a procedure was exported from &quot;project A&quot;, this argument allows you to import it into &quot;project B&quot;, but only if the export used the <code>path</code> option also.</td>
</tr>
<tr>
<td><strong>preserveId</strong></td>
<td>If performing a partial import, using this option preserves the original job ID or workflow ID.</td>
</tr>
</tbody>
</table>

### Positional arguments

- **fileName**

### Response

None or a status OK message.

### ec-perl

**syntax examples:**

```perl
$cmdr->import(<fileName>, {...});

$cmdr->import((file => <localFileName>, ...));
```

**Examples**

```perl
$cmdr->import("/opt/TestProg.xml");

$cmdr->import({file => "c:\r.xml", path => "/projects[Test]"});
```

### ectool

**syntax examples:**

```ectool
ectool import <remoteFileName> ...

ectool import --file <localFileName>
```
**Examples**

```bash
ectool import /mnt/backups/fullBackup.xml

ectool --file "c:\project.xml" --path "/projects[Test]"
```

Back to Top
API Response and Element Glossary

The first part of this help topic lists returned response container elements in alphabetical order. The Contents for each container element lists all or most of the possible returned response elements—both simple and subcontainer elements. Depending on your request, you may not see all elements in your response. If the value of an element is "empty," typically that element is omitted from the response.

**Note:** Elements annotated with an * (asterik) may appear multiple times in a response.

The second part of this help topic is an element glossary for all single or "leaf" elements and subcontainer elements. Click here to go to the glossary or notice that each response element is a link—each response element is linked directly to its glossary entry.

**access**

Contains the set of effective permissions for a user or a group.

Contents:

- changePermissionsPrivilege
- executePrivilege
- modifyPrivilege
- readPrivilege

**aclEntry**

Contains an ACE (access control list entry) on an object for a given principal.

Contents:

- aclEntryId
- changePermissionsPrivilege
- executePrivilege
- modifyPrivilege
- readPrivilege
- principalName
- principalType

**actualParameter**

An actualParameter object provides the value for a parameter, which is passed to a procedure when it is invoked.

Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"—formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
Contents:
- actualParameterId
- actualParameterName
- createTime
- modifyTime
- value

artifact

Contains elements to define the artifact. An artifact is specified by groupId and artifactKey. The name of an artifact is in this form "groupId:artifactKey". An artifact contains a collection of artifactVersions.

Contents:
- artifactId
- artifactKey
- artifactName
- artifactVersionNameTemplate
- createTime
- description
- groupId
- lastModifiedBy
- modifyTime
- owner
- propertySheetId

artifactVersion

A "concrete" version of an artifact that contains a collection of files stored in the artifact repository.

Contents:
- artifactKey
- artifactName
- artifactVersionId
- artifactVersionName
- artifactVersionState
- buildNumber
- createTime
- majorMinorPatch
- modifyTime
- owner
- propertySheetId
- publisherJobId
- publisherJobName
- publisherJobStepId
dependentArtifacts qualifier
description repositoryName
groupId retrievers
lastModifiedBy version

credential
Contains a stored credential. The password is returned for the getFullCredential API only.

Contents:
credentialId
credentialName
createTime
description
lastModifiedBy
modifyTime
owner
password
projectName
propertySheetId
userName
databaseConfiguration
Contain configuration information about communicating with the database used to store server data.

Contents:
batchRequests
batchSize
completeUserName
customDatabaseDialect
customDatabaseDriver
customDatabaseUrl
databaseDialect
databaseDriver
databaseName
databaseType
databaseUrl
hostName
port
statementCacheSize
userName

directoryProvider
Contains information about the configuration used to communicate with an external directory service (LDAP or ActiveDirectory).

Contents:

commonGroupNameAttribute modifyTime
createTime name
description owner
directoryProviderId position
domainName propertySheetId
domainName emailAttribute
enableGroups providerIndex
fullUserNameAttribute providerName
groupBase
groupNameAttribute realm
groupMemberAttributes url
groupMemberFilter useSSL
groupNameAttribute userBase
groupSearchFilter userNameAttribute
lastModifiedBy userSearchFilter
managerDn userSearchSubtree

testDirectoryProvider
Contains the results of testing a directory provider configuration as a list of test result blocks.

Each block contains a result with details about any failures. The findGroupsTest block also includes a list of groups for the test user.

The findUserTest block includes information about the user or users that matched the test user name.

Contents:

findGroupsTest
testResult
details
emailConfig

Contains information about the configuration used to communicate with an email server.

Contents:
  configName
  createTime
  description
  emailConfigId
  emailConfigName
  lastModifiedBy
  mailFrom
  mailHost
  mailPort
  mailProtocol
  mailUser
  modifyTime
  owner
  propertySheetId

emailNotifier

Contains information about an email notifier.
Contents:
condition
cfgName
container
createTime
description
destinations
eemailNotifierId
eventType
formattingTemplate
lastModifiedBy
modifyTime
notifierName
owner
propertySheetId

**formalParameter**

Contains information about a formal parameter.

Contents:
container
createTime
defaultValue
description
expansionDeferred
formalParameterId
formalParameterName
lastModifiedBy
modifyTime
owner
required
type

**gateway**

Contains information about a gateway.

Contents:
group

Contains information about a defined group of users.

Contents:
createTime
groupId
groupName
lastModifiedBy
modifyTime
mutable
owner
propertySheet
propertySheetId
providerName
users

job

Contains information about a running or completed job. Different API calls will result in different subsets of possible properties on the job. Refer to the specific API for details.

Contents:
<table>
<thead>
<tr>
<th><strong>abortedBy</strong></th>
<th><strong>licenseWaitTime</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>abortStatus</strong></td>
<td><strong>liveProcedure</strong></td>
</tr>
<tr>
<td><strong>actualParameters</strong>*</td>
<td><strong>liveSchedule</strong></td>
</tr>
<tr>
<td><strong>callingState</strong></td>
<td><strong>modifyTime</strong></td>
</tr>
<tr>
<td><strong>combinedStatus</strong></td>
<td><strong>outcome</strong></td>
</tr>
<tr>
<td><strong>createTime</strong></td>
<td><strong>owner</strong></td>
</tr>
<tr>
<td><strong>credentialName</strong></td>
<td><strong>priority</strong></td>
</tr>
<tr>
<td><strong>deleted</strong></td>
<td><strong>procedureName</strong></td>
</tr>
<tr>
<td><strong>directoryName</strong></td>
<td><strong>propertyName</strong></td>
</tr>
<tr>
<td><strong>elapsedTime</strong></td>
<td><strong>propertySheet</strong></td>
</tr>
<tr>
<td><strong>errorCode</strong></td>
<td><strong>propertySheetId</strong></td>
</tr>
<tr>
<td><strong>errorMessage</strong></td>
<td><strong>resourceWaitTime</strong></td>
</tr>
<tr>
<td><strong>external</strong></td>
<td><strong>runAsUser</strong></td>
</tr>
<tr>
<td><strong>finish</strong></td>
<td><strong>scheduleName</strong></td>
</tr>
<tr>
<td><strong>jobId</strong></td>
<td><strong>start</strong></td>
</tr>
<tr>
<td><strong>jobName</strong></td>
<td><strong>status</strong></td>
</tr>
<tr>
<td><strong>jobStep</strong>*</td>
<td><strong>steps</strong></td>
</tr>
<tr>
<td><strong>lastModifiedBy</strong></td>
<td><strong>totalWaitTime</strong></td>
</tr>
<tr>
<td><strong>launchedByUser</strong></td>
<td><strong>workspaceWaitTime</strong></td>
</tr>
</tbody>
</table>

---

**jobStep**

Contains information to define or locate a job step. Notice that the calledProcedure element (subcontainer element) can contain multiple jobStep elements.

**Contents:**

<table>
<thead>
<tr>
<th><strong>abortedBy</strong></th>
<th><strong>outcome</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>abortStatus</strong></td>
<td><strong>owner</strong></td>
</tr>
<tr>
<td><strong>actualParameters</strong></td>
<td><strong>parallel</strong></td>
</tr>
<tr>
<td><strong>alwaysRun</strong></td>
<td><strong>postExitCode</strong></td>
</tr>
<tr>
<td><strong>assignedResourceName</strong></td>
<td><strong>postLogFileName</strong></td>
</tr>
<tr>
<td><strong>broadcast</strong></td>
<td><strong>postProcessor</strong></td>
</tr>
<tr>
<td><strong>calledProcedure</strong></td>
<td><strong>precondition</strong></td>
</tr>
<tr>
<td><strong>jobStep</strong>*</td>
<td><strong>procedureName</strong></td>
</tr>
</tbody>
</table>

---

397
license

Contains information to specify the Commander license.

Contents:
createTime
customerName
evaluation
expirationDate
featureName
licenseUsage

Contains information about Commander license usage.

**Note:** Your response will be different depending on how you are licensed for ElectricCommander currently.

Contents:

- concurrentResources
  - inUseHosts
  - inUseProxiedHosts
  - maxHosts
  - maxProxiedHosts
- concurrentUsers*
  - adminLicenseLastUse
  - adminLicenseUser
  - inUseLicenses
  - maxLicenses
  - license*
    - admin
    - expiration
    - lastUse
    - user
- concurrentSteps
  - maxConcurrentSteps
  - runningSteps

logEntry

Contains information about log events generated anywhere in the system.
Contents:
category
container
containerName
deleted
logEntryId
message
principal
severity
subject
subjectName
time

**object**

Primarily, the object element is returned from a `getAccess` API request. If multiple objects are returned, they are presented in an order beginning with the API requested object to the top-level object in the ACL hierarchy. Your object-query response can contain one or more `aclEntry` containers.

Contents:
objectId
objectName
objectType
aclEntry*

**plugin**

Contains elements to define the plugin.

Contents:
author
createTime
description
label
lastModifiedBy
modifyTime
owner
pluginId
pluginKey
pluginName
pluginVersion
project
projectName
promoted
propertySheetId

**procedure**

Contains elements to define the procedure.

Contents:
- attachedCredentials
- createTime
- credentialName
- description
- jobNameTemplate
- lastModifiedBy
- modifyTime
- owner
- procedureId
- procedureName
- projectName
- propertySheetId
- resourceName
- workspaceName

**project**

Contains all elements to define a project.

Contents:
- attachedCredentials
- createTime
- credentialName
- deleted
- description
- lastModifiedBy
- modifyTime
- owner
pluginName
projectId
projectName
propertySheetId
resourceName
workspaceName

**property**

Contains property sheets and various elements, depending on your query.

Contents:
createTime
description
expandable
lastModifiedBy
modifyTime
owner
path
propertyId
propertyName
propertySheet*
propertySheetId
value

**propertySheet**

Contains one or more property elements.

Contents:
createTime
lastModifiedBy
modifyTime
owner
property*
propertySheetId
**repository**

Contains elements to define the artifact repository. The most useful elements in this object are "repositoryName" and "url". Clients publishing/retrieving artifact versions search repositories by name to obtain connection information.

Contents:

- createTime
- description
- lastModifiedBy
- modifyTime
- owner
- propertySheetId
- repositoryDisabled
- repositoryId
- repositoryIndex
- repositoryName
- url
- zoneName

**resource**

Contains elements to define a resource.

Contents:

- agentState
- alive
- code
- details
- message
- pingToken
- protocolVersion
- state
- time
- version
- artifactCacheDirectory
- createTime
- description
- lastRunTime
- modifyTime
- owner
- pools
- port
- propertySheetId
- proxyCustomization
- proxyHostName
- proxyPort
- proxyProtocol
- repositoryNames
- resourceDisabled
- resourceId
exclusiveJobId exclusiveJobName exclusiveJobStepId exclusiveJobStepName gateways hostName hostOS hostPlatform lastModifiedBy

resourcePool

Contains elements to define a resource pool.

Contents:

- autoDelete
- createTime
- description
- lastModifiedBy
- lastResourceUsed
- modifyTime
- orderingFilter
- owner
- propertySheetId
- resourceNames
- resourcePoolDisabled
- resourcePoolId
- resourcePoolName

resourceUsage

Contains information about resource usage. For any step running on a resource, there is a resource usage record containing the ID and name of the job, job step, and resource.

Contents:

- jobId
- jobName
- jobStepId
jobStepName
licenceWaitTime
resourceId
resourceName
resourcePoolId
resourcePoolName
resourceUsageId
resourceWaitTime
waitReason
workspaceWaitTime

**schedule**

Contains all elements to define a schedule.

Contents:

- `actualParameters`
- `attachedCredentials`
- `beginDate`
- `createTime`
- `credentialName`
- `description`
- `endDate`
- `interval`
- `intervalUnits`
- `lastModifiedBy`
- `lastRunTime`
- `misfirePolicy`
- `modifyTime`
- `monthDays`
- `owner`
- `priority`
- `procedureName`
- `projectName`
- `propertySheetId`
- `scheduleDisabled`
- `scheduleId`
- `scheduleName`
- `startTime`
- `stopTime`
- `timeZone`
- `weekDays`

**serverStatus**

Contains elements to determine the status of the server.

Contents:

- `apiMonitor`
- `longestCall`
api
callId
description
elapsedTime
label
remoteAddress
start
userName
mostActiveCalls
totalCallCount
activeCalls
call*
    api
callId
description
elapsedTime
label
remoteAddress
start
userName
recentCalls
call*
    api
callId
description
elapsedTime
label
remoteAddress
start
userName
lastMessage
messages
    message*
serverState
startTime
**serverVersion**

Contains elements to specify the Commander server version.

Contents:

- label
- protocolVersion
- schemaVersion
- version

**state**

Contains elements for a state in a running or completed workflow.

Contents:

- active
- createTime
- description
- errorMessage
- index
- lastModifiedBy
- modifyTime
- owner
- projectName
- propertySheetId
- stateId
- stateName
- subjob
- subprocedure
- subproject
- substartingState
- subworkflow
- subworkflowDefinition
- workflowName

**stateDefinition**

Contains elements for the state definition within a workflow definition.

Contents:
step

Contains elements to specify or define a step.

Contents:

*actualParameters* postLogFileName
alwaysRun postProcessor
attachedCredentials precondition
attachedParameters procedureName
broadcast projectName
command propertySheetId
condition releaseExclusive
createTime releaseMode
credentialName* resourceName
description shell
errorHandling stepId
exclusive stepName
exclusiveMode subprocedure
transition

Contains elements about a transition in a running or completed workflow.

Contents:
- actualParameters
- condition
- createTime
- description
- index
- lastModifiedBy
- modifyTime
- owner
- projectName
- propertySheetId
- stateName
- targetState
- transitionId
- transitionName
- trigger
- workflowName

transitionDefinition

Contains elements about a transition definition within a workflow definition.

Contents:
- actualParameters
- condition
- createTime
- description
- index
Contains elements about a running or completed workflow.

Contents:
- activeState
- callingState
- completed
- createTime
deleted
elapsedTime
finish
lastModifedBy
launchedByUser
liveWorkflowDefinition
modifyTime
owner
projectName
propertySheetId
start
startingState
workflowDefinitionName
workflowId
workflowName

**workflowDefinition**

Contains elements about a workflow definition.

Contents:
createTime
description
lastModifiedBy
modifyTime
owner
projectName
propertySheetId
workflowDefinitionId
workflowDefinitionName
workflowNameTemplate

**workspace**

Contains elements about a workspace.

Contents:
agentDrivePath
agentUncPath
agentUnixPath
createTime
credentialName
description
lastModifiedBy
local
modifyTime
owner
propertySheet
propertySheetId
workspaceDisabled
workspaceId
workspaceName
zoneName

**zone**

Contains elements about a zone.

Contents:
createTime
description
lastModifiedBy
modifyTime
owner
propertySheetId
resources
zoneId
zoneName

**Element Glossary**

The following table lists all simple returned elements, including the element type and its description.

<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortStatus</td>
<td>enum</td>
<td>Possible values are: abort</td>
</tr>
<tr>
<td>abortedBy</td>
<td>string</td>
<td>The name of the user who aborted the job.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>aclEntryId</td>
<td>number</td>
<td>The unique Commander-generated ID for this aclEntry object.</td>
</tr>
<tr>
<td>active</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>activeCalls</td>
<td>subcontainer</td>
<td>A container element within the serverStatus element. activeCall describes an API currently running on the server.</td>
</tr>
<tr>
<td>activeState</td>
<td>string</td>
<td>The name of the activeState on the workflow object.</td>
</tr>
<tr>
<td>actualParameters</td>
<td>propertySheet</td>
<td>An actualParameter object provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from &quot;formal parameters&quot;- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time. For the workflow feature, these are the parameters that were passed when the workflow was started.</td>
</tr>
<tr>
<td>actualParameterId</td>
<td>number</td>
<td>The unique Commander-generated ID for this actual parameter object.</td>
</tr>
<tr>
<td>actualParameterName</td>
<td>string</td>
<td>The name of the parameter. This name is unique within the step, and at run time it matches the name of a formal parameter in the subprocedure.</td>
</tr>
<tr>
<td>admin</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>adminLicenseLastUse</td>
<td>date</td>
<td>The time at which the admin license was last used.</td>
</tr>
<tr>
<td>adminLicenseUser</td>
<td>string</td>
<td>The name of the user who is currently licensed as the &quot;admin&quot; user.</td>
</tr>
<tr>
<td>agentDrivePath</td>
<td>string</td>
<td>Drive-letter-based path used by Windows agents to access the workspace in steps.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>agentUncPath</td>
<td>string</td>
<td>UNC path used by Windows Commander Web servers to access the workspace. The agent uses agentUncPath and agentDrivePath to compute the drive mapping needed for making agentDrivePath valid in the step.</td>
</tr>
<tr>
<td>agentUnixPath</td>
<td>string</td>
<td>UNIX path used by UNIX agents and Linux Commander Web servers to access the workspace.</td>
</tr>
<tr>
<td>agentState</td>
<td>subcontainer</td>
<td>A subcontainer element returned from certain resource queries. agentState returns specific information about an agent, including the state of the agent. Possible values are: unknown</td>
</tr>
<tr>
<td>alive</td>
<td>boolean</td>
<td>Refers to the agent state or status.</td>
</tr>
<tr>
<td>alwaysRun</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>api</td>
<td>string</td>
<td>An element returned on longestCall, activeCall, and recentCall subcontainers of the serverStatus element. api returns the API call (command) that is running or ran on the server.</td>
</tr>
<tr>
<td>apiMonitor</td>
<td></td>
<td>A server object that tracks API active and recent calls, as well as the total number of calls since server startup.</td>
</tr>
<tr>
<td>artifactCacheDirectory</td>
<td>string</td>
<td>The directory on the agent host where retrieved artifacts are stored.</td>
</tr>
<tr>
<td>artifactId</td>
<td>number</td>
<td>The unique Commander-generated ID for this artifact object.</td>
</tr>
<tr>
<td>artifactKey</td>
<td>string</td>
<td>User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td>artifactName</td>
<td>string</td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td>artifactsDirectory</td>
<td>string</td>
<td>The directory in the workspace where you can put files to view, using a report link.</td>
</tr>
<tr>
<td>artifactVersionId</td>
<td>string</td>
<td>The unique Commander-generated ID for this artifact version object.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>name</td>
<td>The name of the artifact version. An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the Commander server interprets either name form correctly.</td>
</tr>
<tr>
<td>artifactVersionNameTemplate</td>
<td>string</td>
<td>A template for the names of artifact versions published to this artifact. Over-rides the <code>globalArtifactVersionNameTemplate</code>. The global setting can be manipulated in the Server Settings page (Administration &gt; Server, select the Settings link).</td>
</tr>
<tr>
<td>artifactVersionState</td>
<td>enum</td>
<td>Possible values are: `available</td>
</tr>
<tr>
<td>assignedResourceName</td>
<td>string</td>
<td>The name of the resource assigned to the step by the step scheduler.</td>
</tr>
<tr>
<td>attachedCredentials</td>
<td>list</td>
<td>The names of the credentials attached to the specified object.</td>
</tr>
<tr>
<td>attachedParameters</td>
<td>string</td>
<td>These are credential parameters that were attached to a step.</td>
</tr>
<tr>
<td>author</td>
<td>string</td>
<td>The author of the plugin.</td>
</tr>
<tr>
<td>autoDelete</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>batchRequests</td>
<td>string</td>
<td>A setting in the database configuration that determines whether or not to batch SQL queries when making a request to the database.</td>
</tr>
<tr>
<td>batchSize</td>
<td>string</td>
<td>The number of objects imported before being committed to the database.</td>
</tr>
<tr>
<td>beginDate</td>
<td>string</td>
<td><code>&lt;yyyy-mm-dd&gt;</code> The date the schedule is set to begin.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>broadcast</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>buildNumber</td>
<td>string</td>
<td>User-defined build number component of the version attribute for the artifact version.</td>
</tr>
<tr>
<td>call</td>
<td>subcontainer</td>
<td>A subcontainer returned on <code>activeCall</code> and <code>recentCall</code> elements returned by the <code>serverStatus</code> API. <code>call</code> contains information specific to each API call on the server.</td>
</tr>
<tr>
<td>callId</td>
<td>number</td>
<td>A unique Commander-generated identifier for this particular call.</td>
</tr>
<tr>
<td>callingState</td>
<td>string</td>
<td>The full property path to the &quot;calling state&quot;, which can appear on subjobs and subworkflows of a workflow.</td>
</tr>
<tr>
<td>calledProcedure</td>
<td>list</td>
<td>A subcontainer element within the <code>jobStep</code> element. The <code>calledProcedure</code> element can contain multiple <code>jobStep</code> elements.</td>
</tr>
<tr>
<td>category</td>
<td>(currently not used)</td>
<td></td>
</tr>
<tr>
<td>changePermissionsPrivilege</td>
<td>enum</td>
<td>Possible values are: allow</td>
</tr>
<tr>
<td>code</td>
<td>enum</td>
<td>Script to execute the functions for a step—passed to the step's shell for execution.</td>
</tr>
<tr>
<td>combinedStatus</td>
<td>enum</td>
<td>More inclusive step status output - this value may combine up to three sub-elements: status</td>
</tr>
<tr>
<td>command</td>
<td>string</td>
<td>The command to run steps - for command steps.</td>
</tr>
<tr>
<td>commonGroupNameAttribute</td>
<td>string</td>
<td>The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider.</td>
</tr>
<tr>
<td>completed</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>completeUserName</td>
<td>string</td>
<td>A SQL server-specific tag that includes the user's name and the user's domain name.</td>
</tr>
<tr>
<td>concurrentResources</td>
<td>object</td>
<td>A subcontainer element that includes information about &quot;in use&quot; and &quot;maximum licensed&quot; hosts and proxied hosts for the licenseUsage API command.</td>
</tr>
<tr>
<td>concurrentSteps</td>
<td>number</td>
<td>The total number of steps running at the same time in the Commander system. This means all steps from all procedures, regardless of how many or how few projects you have created.</td>
</tr>
<tr>
<td>concurrentUsers</td>
<td>object</td>
<td>A subcontainer element that includes information about the admin license, &quot;in use&quot; licenses, and the maximum number of licenses for the licenseUsage API command.</td>
</tr>
<tr>
<td>condition</td>
<td>string</td>
<td><strong>For steps:</strong> If empty or non-zero, the step will run. If set to &quot;0&quot;, the step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps. <strong>For email notifiers:</strong> Mail sent if the condition evaluates to &quot;true&quot;. The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is &quot;false&quot; or &quot;0&quot;. If no condition is specified, the notification is ALWAYS sent.</td>
</tr>
<tr>
<td>configName</td>
<td>string</td>
<td>The name of the configuration.</td>
</tr>
<tr>
<td>container</td>
<td>string</td>
<td>An object ID for a &quot;container&quot; that contains formal parameters. In another context, this is typically the type and name of the workflow or job with a corresponding ID.</td>
</tr>
<tr>
<td>containerName</td>
<td>string</td>
<td>The name of the container.</td>
</tr>
<tr>
<td>createTime</td>
<td>date</td>
<td>The time when this object was created.</td>
</tr>
<tr>
<td>credentialId</td>
<td>number</td>
<td>The unique Commander-generated ID for this credential object.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>credentialName</td>
<td>string</td>
<td>credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. Requires a qualifying project name, absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
<tr>
<td>customDatabaseDialect</td>
<td>string</td>
<td>Class name for the Hibernate dialect. The server chooses an appropriate dialect based on databaseType or this can be part of the custom specification.</td>
</tr>
<tr>
<td>customDatabaseDriver</td>
<td>string</td>
<td>Class name of the JDBC driver. The server will choose an appropriate driver based on databaseType or this can be part of the custom specification.</td>
</tr>
<tr>
<td>customDatabaseUrl</td>
<td>string</td>
<td>The JDBC URL to use. The server will compose an appropriate URL or this can be part of the custom specification.</td>
</tr>
<tr>
<td>customerName</td>
<td>string</td>
<td>The name of a company and/or group name with a company that is using ElectricCommander.</td>
</tr>
<tr>
<td>databaseDialect</td>
<td>string</td>
<td>Class name for the Hibernate dialect (the server chooses an appropriate dialect based on databaseType).</td>
</tr>
<tr>
<td>databaseDriver</td>
<td>string</td>
<td>Class name of the JDBC driver (the server will choose an appropriate driver based on databaseType).</td>
</tr>
<tr>
<td>databaseName</td>
<td>string</td>
<td>The name of the database the Commander server is using.</td>
</tr>
<tr>
<td>databaseType</td>
<td>enum</td>
<td>Possible values are: builtin</td>
</tr>
<tr>
<td>databaseUrl</td>
<td>string</td>
<td>The JDBC URL to use (the server will compose an appropriate URL).</td>
</tr>
<tr>
<td>defaultValue</td>
<td>string</td>
<td>This value is used for the formal parameter if a value is not supplied by the caller.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>delayUntil</td>
<td>date</td>
<td>For a step that was rescheduled due to a resource or workspace problem, this is the next time when the step will be eligible to run.</td>
</tr>
<tr>
<td>deleted</td>
<td>byte</td>
<td>The object was marked for background deletion. Possible values are &quot;0&quot; or &quot;1&quot;. Default is &quot;0&quot; (not set).</td>
</tr>
<tr>
<td>dependentArtifacts</td>
<td>string</td>
<td>A space-separated list of artifacts.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tt&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td>destinations</td>
<td>string</td>
<td>A space-separated list of valid email addresses, email aliases, or Commander user names, or a string subject to property expansion that expands into such a list.</td>
</tr>
<tr>
<td>details</td>
<td>string</td>
<td>A string containing details about agent status.</td>
</tr>
<tr>
<td>directoryName</td>
<td>string</td>
<td>The name of the job's directory within each workspace for a job.</td>
</tr>
<tr>
<td>directoryProviderId</td>
<td>number</td>
<td>The unique Commander-generated ID for this directory provider object.</td>
</tr>
<tr>
<td>domainName</td>
<td>string</td>
<td>The name of the domain from which the Active Directory server(s) are automatically discovered.</td>
</tr>
<tr>
<td>elapsedTime</td>
<td>number</td>
<td>The number of milliseconds between the start and end times for the job or job step - or a workflow.</td>
</tr>
<tr>
<td>email</td>
<td>string</td>
<td>The user's email address.</td>
</tr>
<tr>
<td>emailAttribute</td>
<td>string</td>
<td>The attribute in a user record that contains the user's email address. If the attribute is not specified, the account name and domain name are concatenated to form an email address.</td>
</tr>
<tr>
<td>emailConfigId</td>
<td>number</td>
<td>The unique Commander-generated ID for this email configuration object.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>emailConfigName</td>
<td>string</td>
<td>The name of the email configuration.</td>
</tr>
<tr>
<td>emailNotifierId</td>
<td>number</td>
<td>The unique Commander-generated ID for this email notifier object.</td>
</tr>
<tr>
<td>enableGroups</td>
<td>boolean</td>
<td>Determines whether or not to enable external groups for the directory provider.</td>
</tr>
<tr>
<td>endDate</td>
<td>string</td>
<td>&lt;yyyy-mm-dd&gt; The date this schedule is set to end.</td>
</tr>
<tr>
<td>errorCode</td>
<td>enum</td>
<td>Displays the error code, identifying which error occurred.</td>
</tr>
<tr>
<td>errorHandling</td>
<td>enum</td>
<td>Determines what happens to the procedure if the step fails:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- failProcedure - The current procedure continues, but the overall status is error (default).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- abortProcedure - Aborts the current procedure, but allows already-running steps in the current procedure to complete.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- abortProcedureNow - Aborts the current procedure and terminates running steps in the current procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- abortJob - Aborts the entire job, terminates running steps, but allows alwaysRun steps to run.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- abortJobNow - Aborts the entire job and terminates all running steps, including alwaysRun steps.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ignore - Continues as if the step succeeded.</td>
</tr>
<tr>
<td>errorMessage</td>
<td>string</td>
<td>A description of the error.</td>
</tr>
<tr>
<td>evaluation</td>
<td>boolean</td>
<td>Determines whether or not this license is an evaluation copy only.</td>
</tr>
<tr>
<td>eventType</td>
<td>enum</td>
<td>Possible values are: onCompletion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;onStart&quot; triggers an event when the job or job step begins. &quot;onCompletion&quot; triggers an event when the job finishes, no matter how it finishes. Default is &quot;onCompletion&quot;.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>exclusive</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>exclusiveJobId</td>
<td>number</td>
<td>The ID number of the job that owns this resource, which occurs when one of the job’s steps requests exclusive use of the resource for the duration of the job.</td>
</tr>
<tr>
<td>exclusiveJobName</td>
<td>string</td>
<td>The name of the job that owns this resource, which occurs when one of the job’s steps requests exclusive use of the resource for the duration of the job.</td>
</tr>
<tr>
<td>exclusiveJobStepId</td>
<td>number</td>
<td>The ID number of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.</td>
</tr>
<tr>
<td>exclusiveJobStepName</td>
<td>name</td>
<td>The name of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.</td>
</tr>
<tr>
<td>exclusiveMode</td>
<td>enum</td>
<td>Possible values are: none</td>
</tr>
<tr>
<td>executePrivilege</td>
<td>enum</td>
<td>Possible values are: allow</td>
</tr>
<tr>
<td>exitCode</td>
<td>number</td>
<td>The step's exit code.</td>
</tr>
<tr>
<td>expandable</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>expansionDeferred</td>
<td>boolean</td>
<td>`&lt;Boolean flag -0</td>
</tr>
<tr>
<td>expiration</td>
<td>date</td>
<td>The date when a user license expires.</td>
</tr>
<tr>
<td>expirationDate</td>
<td>date</td>
<td>The date when a license expires.</td>
</tr>
</tbody>
</table>
| external           | boolean   | `<Boolean flag -0|1|true|false>` If "true," this job is external. For more information about external jobs, see the API Commands - Job Management Help topic.
<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureName</td>
<td>string</td>
<td>The name of the licensed feature. Possible features include: Server</td>
</tr>
<tr>
<td>findGroupsTest</td>
<td>subcontainer</td>
<td>For the testDirectoryProvider API, this element provides information on which groups the user is a member.</td>
</tr>
<tr>
<td>findUserTest</td>
<td>subcontainer</td>
<td>For the testDirectoryProvider API, this element contains specific information about the user.</td>
</tr>
<tr>
<td>finish</td>
<td>date</td>
<td>The time the job or workflow completed.</td>
</tr>
<tr>
<td>formalParameterId</td>
<td>number</td>
<td>The formal parameter's ID.</td>
</tr>
<tr>
<td>formalParameterName</td>
<td>string</td>
<td>The name of the procedure's parameter, containing a credential reference.</td>
</tr>
<tr>
<td>formalParameters</td>
<td>string</td>
<td>The parameters that must be supplied when entering the state (similar to formal parameters on a procedure).</td>
</tr>
<tr>
<td>formattingTemplate</td>
<td>string</td>
<td>Specifies a template for formatting email messages when an event [notification] is triggered by the emailNotifier.</td>
</tr>
<tr>
<td>fullUserName</td>
<td>string</td>
<td>The user's full name - not his or her nickname.</td>
</tr>
<tr>
<td>fullUserNameAttribute</td>
<td>string</td>
<td>The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.</td>
</tr>
<tr>
<td>gatewayDisabled</td>
<td>boolean</td>
<td>`&lt;Boolean flag -0</td>
</tr>
<tr>
<td>gatewayId</td>
<td>number</td>
<td>The Commander-generated ID number for this gateway.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>string</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>gateways</td>
<td>list</td>
<td>A space-separated list of gateway names.</td>
</tr>
<tr>
<td>gracePeriod</td>
<td>number</td>
<td>The number of days available after the Commander license expires.</td>
</tr>
<tr>
<td>groupBase</td>
<td>string</td>
<td>This string is prepended to the basedn to construct the directory DN that contains group records.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>groupId</td>
<td>number</td>
<td>The unique Commander-generated group ID. For Artifact Management: A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td>groupList</td>
<td>list</td>
<td>For the <code>testDirectoryProvider</code> API, this element contains zero or more groups returned after querying existing groups known to the directory provider.</td>
</tr>
<tr>
<td>groupMemberAttributes</td>
<td>string</td>
<td>A comma-separated attribute name list that identifies a group member. Most LDAP configurations only specify a single value, but if there is a mixture of POSIX and LDAP style groups in the directory, multiple attributes might be required.</td>
</tr>
</tbody>
</table>
| groupMemberFilter   | string     | This LDAP query is performed in the groups directory context to identify groups containing a specific user as a member. Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and `memberOfNames` or `uniqueGroupOfNames` records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: 
                             
<p>| groupName           | string     | The full name of a group. For Active Directory and LDAP, this is a full DN.                                                                       |
| groupNameAttribute  | string     | The group record attribute that contains the name of the group.                                                                                      |
| groups              | list       | A space-separated list of group names.                                                                                                              |
| groupSearchFilter   | string     | The LDAP query performed in the context of the groups directory to enumerate group records.                                                           |
| groupSettingsId     | number     | The unique Commander-generated ID for this group settings object.                                                                                     |
| hostName            | string     | The computer name or IP address for the machine containing the Commander server or agent.                                                             |</p>
<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostName1</td>
<td>string</td>
<td>For gateways: The name Resource 2 uses to communicate with Resource 1. If &quot;blank&quot;, the Agent Host Name attribute in Resource 1’s definition is used at runtime.</td>
</tr>
<tr>
<td>hostName2</td>
<td>string</td>
<td>For gateways: The name Resource 1 uses to communicate with Resource 2. If &quot;blank&quot;, the Agent Host Name attribute in Resource 2’s definition is used at runtime.</td>
</tr>
<tr>
<td>hostOS</td>
<td>string</td>
<td>The full name of the host operating system, plus its version. However, if this host is a proxy, the value is &quot;proxied&quot;.</td>
</tr>
<tr>
<td>hostPlatform</td>
<td>string</td>
<td>Examples for &quot;platform&quot; are: Windows, Linux, HPUX, and so on. However, if this host is a proxy, the value is &quot;proxied&quot;.</td>
</tr>
<tr>
<td>index</td>
<td>number</td>
<td>The numeric index of the transition that indicates its order in the list of transitions in a state definition.</td>
</tr>
<tr>
<td>interval</td>
<td>string</td>
<td>The repeat interval for starting new jobs.</td>
</tr>
<tr>
<td>intervalUnits</td>
<td>enum</td>
<td>Possible values are: hours</td>
</tr>
<tr>
<td>inUseHosts</td>
<td>number</td>
<td>The number of hosts (agents) currently in use.</td>
</tr>
<tr>
<td>inUseLicenses</td>
<td>number</td>
<td>The number of user licenses currently in use.</td>
</tr>
<tr>
<td>inUseProxiedHosts</td>
<td>number</td>
<td>The number of proxy target hosts currently in use.</td>
</tr>
<tr>
<td>jobId</td>
<td>number</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobName</td>
<td>string</td>
<td>The name of the job.</td>
</tr>
<tr>
<td>jobNameTemplate</td>
<td>string</td>
<td>Template used to determine the default name of jobs launched from a procedure.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>number</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>jobStepName</td>
<td>string</td>
<td>The name of the job step.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>label</td>
<td>string</td>
<td>A name used by a plugin for display in a list, or this may represent context-specific info about an API call—not all API calls return a &quot;label&quot; tag.</td>
</tr>
<tr>
<td>lastMessage</td>
<td>string</td>
<td>Element returned by the <code>serverStatus</code> API showing the last message the server received.</td>
</tr>
<tr>
<td>lastModifiedBy</td>
<td>string</td>
<td>Shows who (generally a user name) last modified the object.</td>
</tr>
<tr>
<td>lastResourceUsed</td>
<td>string</td>
<td>The name of the most recently used resource from the pool.</td>
</tr>
<tr>
<td>lastRunTime</td>
<td>date</td>
<td>The last time a job was launched by a schedule. -or- In a resource response, this is the most recent time that a job step ran on the resource.</td>
</tr>
<tr>
<td>lastUse</td>
<td></td>
<td>Returned element in the <code>concurrentUsers</code> subcontainer (for the <code>licenseUsage</code> API), providing the last time a specific user accessed Commander.</td>
</tr>
<tr>
<td>launchedByUser</td>
<td>string</td>
<td>The name of the user or project principal that explicitly launched the job. This property is blank when the job is launched by a schedule.</td>
</tr>
<tr>
<td>licenseId</td>
<td>number</td>
<td>The unique Commander-generated ID for this license.</td>
</tr>
<tr>
<td>licenseWaitTime</td>
<td></td>
<td>The amount of time a job step was stalled waiting for an available license. On a job, this is the sum of license wait for all job steps.</td>
</tr>
<tr>
<td>liveProcedure</td>
<td>string</td>
<td>Shows the current procedure name for the procedure step from which the job or job step was created – if the procedure step was renamed since the job or job step was launched, this is the procedure step’s new name, and if the procedure step was deleted, this will be null.</td>
</tr>
<tr>
<td>liveProcedureStep</td>
<td>string</td>
<td>Shows the current procedure step name for the procedure step from which the job step was created – if the procedure step was renamed since the job was launched, this is the procedure step’s new name, and if the procedure step was deleted, this will be null.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>liveSchedule</td>
<td>string</td>
<td>Shows the current schedule name for the procedure step from which the job was created – if the schedule was renamed since the job was launched, this is the schedule’s new name, and if the schedule was deleted, this will be null.</td>
</tr>
<tr>
<td>liveWorkflowDefinition</td>
<td>string</td>
<td>Shows the current workflow definition name for the workflow definition from which the workflow was created – if the workflow definition was renamed since the workflow was launched, this is the workflow definition’s new name, and if the workflow definition was deleted, this will be null.</td>
</tr>
<tr>
<td>local</td>
<td>boolean</td>
<td>&lt;Boolean flag -0</td>
</tr>
<tr>
<td>logEntryId</td>
<td>number</td>
<td>The Commander-generated ID number for the log entry record.</td>
</tr>
<tr>
<td>logFileName</td>
<td>string</td>
<td>A custom log file name produced by running the step. By default, ElectricCommander assigns a unique name for this file.</td>
</tr>
<tr>
<td>longestCall</td>
<td>string</td>
<td>Provides the API call that took the longest time.</td>
</tr>
<tr>
<td>mailFrom</td>
<td>string</td>
<td>The email address used as the email sender address for notifications.</td>
</tr>
<tr>
<td>mailHost</td>
<td>string</td>
<td>The name of the email server host.</td>
</tr>
<tr>
<td>mailPort</td>
<td>number</td>
<td>The port number for the mail server, but may not need to be specified. The protocol software determines the default value (25 for SMTP and 465 for SSMTPT). Specify a value for this argument when a non-default port is used.</td>
</tr>
<tr>
<td>mailProtocol</td>
<td>string</td>
<td>This is either SSMTPT or SMTP (not case-sensitive). The default is SMTP.</td>
</tr>
<tr>
<td>mailUser</td>
<td>string</td>
<td>This can be an individual or a generic name like &quot;Commander&quot; - name of the email user on whose behalf Commander sends email notifications.</td>
</tr>
<tr>
<td>majorMinorPatch</td>
<td>string</td>
<td>major.minor.patch component of the version attribute for the artifact.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>managerDn</td>
<td>string</td>
<td>The name of a user who has read-only access to the LDAP or Active Directory server. Typically a DN (distinguished name). A simple name may be used when the Active Directory server's URL is being auto-discovered via DNS. <strong>Note:</strong> This user does not need to be an admin user with modify privileges.</td>
</tr>
<tr>
<td>maxConcurrentSteps</td>
<td>number</td>
<td>The maximum number of steps that can run at the same time per the provisions of your Commander license.</td>
</tr>
<tr>
<td>maxHosts</td>
<td>number</td>
<td>The maximum number of hosts licensed for resource use.</td>
</tr>
<tr>
<td>maxLicenses</td>
<td>number</td>
<td>The maximum number of licenses available for users.</td>
</tr>
<tr>
<td>maxProxiedHosts</td>
<td>number</td>
<td>The maximum number of available licenses for proxy hosts.</td>
</tr>
<tr>
<td>message</td>
<td>string</td>
<td>A user-readable diagnostic message associated with an error.</td>
</tr>
<tr>
<td>messages</td>
<td>list</td>
<td>Multiple error or diagnostic messages.</td>
</tr>
</tbody>
</table>
| misfirePolicy     | enum    | Possible values are: ignore | run once  
A schedule may not fire at the allotted time because a prior job is still running, the server is running low on resources and there is a delay, or the server is down.  
When the underlying issue is resolved, the server will schedule the next job at the next regularly scheduled time slot if the policy is 'ignore', otherwise it will run the job immediately. Defaults to "ignore". |
<p>| modifyPrivilege   | enum    | Possible values are: allow | deny | inherit  |
| modifyTime        | date    | The time when the object was last modified.                                                                                                                                                                         |
| monthDays         | string  | Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space.                                                                                       |
| mostActiveCalls   | number  | The number of most active API calls since server startup.                                                                                                                                                          |</p>
<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutable</td>
<td>boolean</td>
<td>If &quot;true,&quot; the member list of this group is editable within Commander via the web UI or the modifyGroup API.</td>
</tr>
<tr>
<td>name</td>
<td>string</td>
<td>The name of the directory provider.</td>
</tr>
<tr>
<td>notifierName</td>
<td>string</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>number</td>
<td>An object identifier returned by findObjects and getObjects. This value is a &quot;handle&quot; only for passing to API commands. The internal structure of this value is subject to change - do not parse this value.</td>
</tr>
<tr>
<td>objectName</td>
<td>string</td>
<td>The name of the object.</td>
</tr>
<tr>
<td>objectType</td>
<td>enum</td>
<td>The type of object being described, for example: project, procedure, step, and so on.</td>
</tr>
<tr>
<td>orderingFilter</td>
<td>string</td>
<td>A Javascript block invoked when scheduling resources for a pool. Note: A Javascript block is not required unless you need to override the default resource ordering behavior.</td>
</tr>
<tr>
<td>outcome</td>
<td>enum</td>
<td>Possible values for outcome: success - The job finished successfully. warning - The job completed with no errors, but encountered some suspicious conditions. error - The job has finished execution with errors.</td>
</tr>
<tr>
<td>owner</td>
<td>string</td>
<td>The person (user name) who created the object.</td>
</tr>
<tr>
<td>parallel</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>password</td>
<td>string</td>
<td>The password matching the specified user name.</td>
</tr>
<tr>
<td>path</td>
<td>string</td>
<td>The property path that specifies the object to use.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
<td>-------------------</td>
</tr>
<tr>
<td>pingToken</td>
<td>number</td>
<td>Every time an agent starts, a unique pingToken value is generated. The server uses the pingToken value to determine agent restarts by noticing the values before and after a restart.</td>
</tr>
<tr>
<td>pluginId</td>
<td>number</td>
<td>The unique Commander-generated ID for the plugin object.</td>
</tr>
<tr>
<td>pluginKey</td>
<td>string</td>
<td>The name of the plugin as displayed on the Commander Plugin Manager web page.</td>
</tr>
<tr>
<td>pluginName</td>
<td>string</td>
<td>The name of the plugin - the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.</td>
</tr>
<tr>
<td>pluginVersion</td>
<td>string</td>
<td>The version of the plugin being described.</td>
</tr>
<tr>
<td>pools</td>
<td>list</td>
<td>A space-separated list of one or more pool names where this resource is a member. Steps defined to run on a resource pool will run on any available member (resource) in the pool.</td>
</tr>
<tr>
<td>port</td>
<td>number</td>
<td>If a port number is not specified, the default Commander port is used. For a proxy resource, this is the port number for the service running on the proxy target that will run commands on behalf of the ElectricCommander agent. For ssh, the default is 22.</td>
</tr>
<tr>
<td>port1</td>
<td>number</td>
<td>The port number used by Gateway Resource1 - default is to the port number used by the resource.</td>
</tr>
<tr>
<td>port2</td>
<td>number</td>
<td>The port number used by Gateway Resource2 - default is to the port number used by the resource.</td>
</tr>
<tr>
<td>position</td>
<td>number</td>
<td>Used to reorder a Commander object. For example, if reordering directory providers: the provider name is moved to a position just before this provider. &quot;Blank&quot; means move the provider to the end of the provider list.</td>
</tr>
<tr>
<td>postExitCode</td>
<td>number</td>
<td>The step's post processor exit code.</td>
</tr>
<tr>
<td>postLogFileName</td>
<td>string</td>
<td>The log file name produced by this step's post processor.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>postProcessor</td>
<td>string</td>
<td>This program looks at the step output to find errors and warnings. Commander includes a customizable program called “postp” for this purpose.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The value for postProcessor is a command string for invoking a post-processor program in the platform shell for the resource (cmd for Windows, sh for UNIX).</td>
</tr>
<tr>
<td>precondition</td>
<td>string</td>
<td>Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a precondition is evaluated. A precondition is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</td>
</tr>
<tr>
<td>principal</td>
<td>string</td>
<td>The user or project principal from the session that was active when the event occurred.</td>
</tr>
<tr>
<td>principalName</td>
<td>string</td>
<td>This is either a user or a group name.</td>
</tr>
<tr>
<td>principalType</td>
<td>enum</td>
<td>Possible values are: group</td>
</tr>
<tr>
<td>priority</td>
<td>enum</td>
<td>Possible values are: low</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Priorities take effect when two or more job steps in different jobs are waiting for the same resource.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>When the resource is available, it will be used by the job step that belongs to the job with the highest priority.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number.</td>
</tr>
<tr>
<td>procedureId</td>
<td>number</td>
<td>The unique Commander-generated procedure ID.</td>
</tr>
<tr>
<td>procedureName</td>
<td>string</td>
<td>The name of the procedure - may be a path to the procedure.</td>
</tr>
<tr>
<td>productName</td>
<td>string</td>
<td>The name of the product with the licensed feature. Possible products include: ElectricCommander</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>project</td>
<td>name</td>
<td>The name of the project associated with the plugin.</td>
</tr>
<tr>
<td>projectId</td>
<td>number</td>
<td>The unique Commander-generated project ID.</td>
</tr>
<tr>
<td>projectName</td>
<td>string</td>
<td>The name of the project - may be a path. The project name is ignored for credentials, procedure, steps, and schedules if it is specified as a path.</td>
</tr>
<tr>
<td>promoted</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>propertyId</td>
<td>number</td>
<td>The unique Commander-generated property ID.</td>
</tr>
<tr>
<td>propertyName</td>
<td>string</td>
<td>The name of the property. It may be a relative or absolute property path, including &quot;my&quot; paths such as &quot;/myProject/propl&quot;.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>number</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>protocolVersion</td>
<td>string</td>
<td>The server API protocol version. For example, the server accepts messages from ectool and ec-perl.</td>
</tr>
<tr>
<td>providerIndex</td>
<td>number</td>
<td>The index that specifies the search order across multiple directory providers. For example: 2 LDAP providers, one with index &quot;0&quot; and one with index &quot;1&quot; means the providers will be searched in that numerical order.</td>
</tr>
<tr>
<td>providerName</td>
<td>string</td>
<td>The LDAP or Active Directory provider name.</td>
</tr>
<tr>
<td>providerType</td>
<td>enum</td>
<td>Possible values are: ldap</td>
</tr>
<tr>
<td>proxyCustomization</td>
<td>string</td>
<td>Perl code customizing how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources.</td>
</tr>
<tr>
<td>proxyHostName</td>
<td>string</td>
<td>The name or IP address of the computer containing the ElectricCommander Agent used for a proxy resource.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>proxyPort</td>
<td>number</td>
<td>The ElectricCommander agent port number for a proxy resource.</td>
</tr>
<tr>
<td>proxyProtocol</td>
<td>string</td>
<td>Protocol for communicating with the proxy target. Defaults to <code>ssh</code>. (This argument is not exposed in the ElectricCommander Web Interface at this time.)</td>
</tr>
<tr>
<td>publisherJobId</td>
<td>number</td>
<td>The Commander-generated ID for the job that published the artifact version.</td>
</tr>
<tr>
<td>publisherJobName</td>
<td>name</td>
<td>The name of the job that published the artifact version.</td>
</tr>
<tr>
<td>publisherJobStepId</td>
<td>number</td>
<td>The Commander-generated ID for the job step that published the artifact version.</td>
</tr>
<tr>
<td>qualifier</td>
<td>string</td>
<td>User-defined qualifier component of the version attribute for the artifact.</td>
</tr>
<tr>
<td>readPrivilege</td>
<td>enum</td>
<td>Possible values are: <code>allow</code>, <code>deny</code>, <code>inherit</code>.</td>
</tr>
<tr>
<td>realm</td>
<td>string</td>
<td>The realm of the LDAP directory provider—used to create unique user names when there are multiple providers.</td>
</tr>
<tr>
<td>recentCall</td>
<td>subcontainer</td>
<td>A subcontainer element on the serverStatus API - a call no longer active (completed). The API monitor keeps track of the 10 most recent calls.</td>
</tr>
<tr>
<td>releaseExclusive</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>releaseMode</td>
<td>string</td>
<td>Possible values are: <code>none</code>, <code>release</code>, <code>releaseToJob</code></td>
</tr>
<tr>
<td>remoteAddress</td>
<td>string</td>
<td>Generally a combined IP address plus a port specification - used when the agent is talking to the server or to show where the request to the server originated.</td>
</tr>
<tr>
<td>repositoryDisabled</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>repositoryId</td>
<td>number</td>
<td>The Commander-generated ID for the artifact repository.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>repositoryIndex</td>
<td>integer</td>
<td>The order of the repository within a list of repositories.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>string</td>
<td>The name of the artifact repository.</td>
</tr>
<tr>
<td>repositoryNames</td>
<td>list</td>
<td>A list of one or more repository server names—each repository name listed on a &quot;new line&quot;.</td>
</tr>
<tr>
<td>required</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>resourceDisabled</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>resourceId</td>
<td>number</td>
<td>The unique Commander-generated ID for this resource.</td>
</tr>
<tr>
<td>resourceName1</td>
<td>string</td>
<td>The name for the first of two resources required to create a gateway. &quot;Spaces&quot; are NOT allowed in a resource name.</td>
</tr>
<tr>
<td>resourceName2</td>
<td>string</td>
<td>The name for the second of two resources required to create a gateway. &quot;Spaces&quot; are NOT allowed in a resource name.</td>
</tr>
<tr>
<td>resourceName</td>
<td>string</td>
<td>The name of a resource.</td>
</tr>
<tr>
<td>resourceNames</td>
<td>string</td>
<td>A list of strings that refer to resources that belong to the pool. Names that do not refer to existing resources are ignored.</td>
</tr>
<tr>
<td>resourcePoolDisabled</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>resourcePoolId</td>
<td>number</td>
<td>The unique ID number for a resource pool.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>name</td>
<td>The name of the resource pool.</td>
</tr>
<tr>
<td>resources</td>
<td>string</td>
<td>A space-separated list of resource names.</td>
</tr>
<tr>
<td>resourceUsageId</td>
<td>number</td>
<td>The unique ID number of the resource usage record.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>resourceWaitTime</td>
<td>string</td>
<td>The amount of time a job step waited for a resource to become available. On a job, this is the sum of time all job steps waited for resource availability. This could indicate that eligible resources for the step have reached their step limit, are in-use but the step requires a resource exclusively, or resources are down.</td>
</tr>
<tr>
<td>retries</td>
<td>number</td>
<td>The number of attempts to write to the step log in the workspace. In a running step, this is the number of retries attempted up to this point. The most common reason for step retries is if the workspace for the step was unavailable.</td>
</tr>
<tr>
<td>retrievers</td>
<td>list</td>
<td>A collection of retrieve elements that can contain a jobName, jobId, and/or a jobStepId element(s).</td>
</tr>
<tr>
<td>runAsUser</td>
<td>string</td>
<td>The name of the user being impersonated in this job.</td>
</tr>
<tr>
<td>runnable</td>
<td>date</td>
<td>The time when the step became runnable.</td>
</tr>
<tr>
<td>runningSteps</td>
<td>number</td>
<td>The number of steps running at the same time.</td>
</tr>
<tr>
<td>runtime</td>
<td>number</td>
<td>The number of milliseconds the step command spent running on a resource.</td>
</tr>
<tr>
<td>scheduleDisabled</td>
<td>boolean</td>
<td>Boolean flag - 0</td>
</tr>
<tr>
<td>scheduleId</td>
<td>number</td>
<td>The unique Commander-generated ID for the schedule.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>string</td>
<td>The name of the schedule - may be a path to the schedule.</td>
</tr>
<tr>
<td>schemaVersion</td>
<td>number</td>
<td>The Commander server's database schema version.</td>
</tr>
<tr>
<td>serverState</td>
<td>enum</td>
<td>Possible values are: bootstrap, databaseConfiguration, databaseConnection, databaseSchema, running, failed, stopping, importFailed</td>
</tr>
<tr>
<td>severity</td>
<td>enum</td>
<td>Possible values are: INFO</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>shell</td>
<td>string</td>
<td>Where shell is the name of a program used to execute commands contained in the &quot;command&quot; field. Normally, this file is a command shell, but it could be any other command line program. The default is &quot;cmd /q /c&quot; for a Windows agent and &quot;sh -e&quot; for a UNIX agent. This is applicable to command steps only.</td>
</tr>
<tr>
<td>signature</td>
<td>string</td>
<td>The digital signature on this license.</td>
</tr>
<tr>
<td>start</td>
<td>date</td>
<td>The time this job or workflow began executing.</td>
</tr>
<tr>
<td>startable</td>
<td>boolean</td>
<td>&quot;True&quot; means this state definition can be the initial state of an instantiated workflow.</td>
</tr>
<tr>
<td>startingState</td>
<td>string</td>
<td>The initial state of the workflow.</td>
</tr>
<tr>
<td>startTime</td>
<td>string</td>
<td>Formatted hh:mm, using the 24-hour clock. Using this schedule, Commander starts creating jobs at this time on the specified days.</td>
</tr>
<tr>
<td>stateDefinitionId</td>
<td>number</td>
<td>The unique Commander-generated ID for this state definition object.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>string</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateId</td>
<td>number</td>
<td>The unique Commander-generated ID for this state object.</td>
</tr>
<tr>
<td>statementCacheSize</td>
<td>string</td>
<td>The number of MS SQL statements cached in the database.</td>
</tr>
<tr>
<td>stateName</td>
<td>string</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>status</td>
<td>enum</td>
<td>Possible values for status: pending - The job is not yet runnable—it is waiting for other steps to complete first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>runnable - The job is ready to run, but it is waiting for a resource to become available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>running - The job is assigned to a resource and is executing the step command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>completed - The job finished executing.</td>
</tr>
<tr>
<td>stepCount</td>
<td>number</td>
<td>The number of executing steps on this resource.</td>
</tr>
<tr>
<td>stepErrorCode</td>
<td>enum</td>
<td>Agent error messages.</td>
</tr>
<tr>
<td>stepId</td>
<td>number</td>
<td>The unique Commander-generated ID for the step.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
<td>-------------------</td>
</tr>
<tr>
<td>stepLimit</td>
<td>number</td>
<td>The number of steps that can run on the resource at one time. (Previously setting the limit to 1 enforces serial access to the resource.)</td>
</tr>
<tr>
<td>stepName</td>
<td>string</td>
<td>The name of the step - may be a path to the step.</td>
</tr>
<tr>
<td>steps</td>
<td></td>
<td>The list or number of steps in a job.</td>
</tr>
<tr>
<td>stopTime</td>
<td>string</td>
<td>Formatted <code>hh:mm</code>, using the 24-hour clock. ElectricCommander stops creating new jobs at this time, but a job in progress will continue to run. If <code>stopTime</code> is not specified, ElectricCommander creates one job only on each specified day.</td>
</tr>
<tr>
<td>subject</td>
<td>string</td>
<td>Refers to the object the event concerns (similar to <code>container</code>).</td>
</tr>
<tr>
<td>subjectName</td>
<td>string</td>
<td>The name of the subject/object.</td>
</tr>
<tr>
<td>subjob</td>
<td>string</td>
<td>The name of the subjob.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>string</td>
<td>The name of the nested procedure called when a step runs. If a subprocedure is specified, <code>command</code> or <code>commandFile</code> options are not necessary.</td>
</tr>
<tr>
<td>subproject</td>
<td>string</td>
<td>If a subprocedure argument was used, this is the name of the project where that subprocedure is found. By default, the current project is used.</td>
</tr>
<tr>
<td>substartingState</td>
<td>string</td>
<td>Name of the starting state for the workflow launched when the state is entered.</td>
</tr>
<tr>
<td>subworkflow</td>
<td>string</td>
<td>The name of the subworkflow.</td>
</tr>
<tr>
<td>subworkflowDefinition</td>
<td>string</td>
<td>The name of the subworkflow definition.</td>
</tr>
<tr>
<td>targetState</td>
<td>string</td>
<td>The target state for the transition definition.</td>
</tr>
<tr>
<td>testResult</td>
<td>enum</td>
<td>Possible values are: `success</td>
</tr>
<tr>
<td>time</td>
<td>date</td>
<td>The time of day to invoke this schedule's procedure (24-hour clock, for example, 17:00). For a <code>logEntry</code> response, <code>time</code> indicates the time at which data was written to the log.</td>
</tr>
</tbody>
</table>
### Returned element | Type | Description/Value
---|---|---
timeLimit | number | The maximum length of time the step is allowed to run. After the time specified, the step will be aborted. The time limit is specified in units that can be hours, minutes, or seconds.
timeLimitUnits | enum | Possible values are: hours|minutes|seconds
timeout | number | Specifies the timeout for the element flag. The default value is 120 seconds.
timeZone | string | The time zone specified to use for this schedule (Java-compatible string).
totalCallCount | number | The total number of API calls to the server since startup.
totalWaitTime | | On a job, this is the sum of total time all job steps waited for license, resource, and/or workspace availability.
transitionDefinitionId | number | The unique Commander-generated ID for this transition definition.
transitionDefinitionName | string | The name of the transition definition.
transitionId | number | The unique Commander-generated ID for this transition object.
transitionName | string | The name of the transition.
trigger | enum | Possible values are: onEnter|onStart|onCompletion|manual
trusted | boolean | `<Boolean flag - 0|1|true|false> if "true", the resource is trusted. A trusted agent is one that has been "certificate verified."

Agents can be either trusted or untrusted:
- trusted - the Commander server verifies the agent's identity using SSL certificate verification.
- untrusted - the Commander server does not verify agent identity. Potentially, an untrusted agent is a security risk.
<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>string</td>
<td>The &quot;type&quot; is any string value. Used primarily by the web interface to represent custom form elements. However, if &quot;credential&quot; is the string value, the server will expect a credential as the parameter value.</td>
</tr>
<tr>
<td>url</td>
<td>string</td>
<td>For directory providers: The server URL is in the form protocol://host:port/basedn. Protocol is either ldap or ldaps (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for ldap, 636 for ldaps). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a <code>dc=</code> and separated by commas. <strong>Note:</strong> Spaces in the <code>basedn</code> must be URL encoded (%20). For artifact repositories: The server URL is in the form protocol://host:port/. Typically, the repository server is configured to listen on port 8200 for https requests, so a typical URL looks like <a href="https://host:8200/">https://host:8200/</a>.</td>
</tr>
<tr>
<td>userAuthenticationTest</td>
<td>subcontainer</td>
<td>For the <code>testDirectoryProvider</code> API, this element authenticates the user.</td>
</tr>
<tr>
<td>userBase</td>
<td>string</td>
<td>The string prepended to the <code>basedn</code> to construct the directory DN that contains user records.</td>
</tr>
<tr>
<td>userId</td>
<td>number</td>
<td>The unique Commander-generated ID for the user.</td>
</tr>
<tr>
<td>userInfo</td>
<td></td>
<td><code>findUserTest</code> container element includes a <code>userList</code> subcontainer that may include multiple <code>userInfo</code> tags, each of which describes a user (including full name, email address, and provider name).</td>
</tr>
<tr>
<td>userList</td>
<td>list</td>
<td><code>findUserTest</code> container element includes a <code>userList</code> subcontainer that may include one or more <code>userInfo</code> tags.</td>
</tr>
<tr>
<td>userName</td>
<td>string</td>
<td>The full name of the user. For Active Directory and LDAP, the name may be <code>user@domain</code>.</td>
</tr>
<tr>
<td>userNameAttribute</td>
<td>string</td>
<td>The attribute in a user record that contains the user's account name.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>userSearchFilter</td>
<td>string</td>
<td>The LDAP query performed in the context of the user directory to search for a user by account name. The string &quot;{0}&quot; is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.</td>
</tr>
<tr>
<td>userSearchSubtree</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>userSettingsId</td>
<td>number</td>
<td>The unique Commander-generated ID for the user settings.</td>
</tr>
<tr>
<td>useSSL</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>For a string property, this is the value of the property. For a sheet property, this argument is invalid.</td>
</tr>
<tr>
<td>version</td>
<td>string</td>
<td>For plugin versions, the value is represented in the form: major.minor. For artifact versions, the value is represented in the form: major.minor.patch-qualifier-buildNumber</td>
</tr>
<tr>
<td>waitReason</td>
<td>string</td>
<td>Possible values are: license, resource, or workspace. Generally, this objects are unavailable, causing a longer wait time for availability.</td>
</tr>
<tr>
<td>waitTime</td>
<td>number</td>
<td>The number of milliseconds the step spent between runnable and running (for example, waiting for a resource).</td>
</tr>
<tr>
<td>weekDays</td>
<td>string</td>
<td>Restricts the schedule to specified days of the week. Days of the week are separated by spaces. English names &quot;Monday&quot;, &quot;Tuesday&quot;, and so on.</td>
</tr>
<tr>
<td>workflowDefinitionId</td>
<td>number</td>
<td>The unique Commander-generated ID for this workflow definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>string</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>workflowId</td>
<td>number</td>
<td>The unique Commander-generated ID for this workflow object.</td>
</tr>
<tr>
<td>workflowName</td>
<td>string</td>
<td>The name of this workflow.</td>
</tr>
<tr>
<td>workflowNameTemplate</td>
<td>string</td>
<td>Template used to determine the default names for workflows launched from a workflow definition.</td>
</tr>
<tr>
<td>workingDirectory</td>
<td>string</td>
<td>The Commander agent sets this directory as the “current working directory,” when running the command contained in the step. If no working directory is specified in the step, Commander uses the directory it created for the job in the Commander workspace as the working directory. <strong>Note:</strong> If running a step on a proxy resource, this directory must exist on the proxy target.</td>
</tr>
<tr>
<td>workspaceDisabled</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workspaceId</td>
<td>number</td>
<td>The unique Commander-generated ID for the workspace.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>string</td>
<td>The name of the workspace.</td>
</tr>
<tr>
<td>workspaceWaitTime</td>
<td></td>
<td>The total time a job step waited for workspace availability. On a job, this is the sum of time all job steps waited for workspace availability.</td>
</tr>
<tr>
<td>zoneId</td>
<td>number</td>
<td>The Commander-generated ID for this zone.</td>
</tr>
<tr>
<td>zoneName</td>
<td>string</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>
# ElectricFlow Glossary

This glossary is a reference topic containing short descriptions for ElectricFlow objects, terms, and concepts. Links to one or more related help topics for a particular "term" are available at the end of most descriptions.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access control</td>
<td>An ACL determines if a particular user can perform a particular operation on a specified object. The list contains access control entries (ACE), each of which specifies a user or group and indicates whether certain operations are allowed or denied for that user or group. Using access control provides security for Commander system use. See the Access Control topic for more information.</td>
</tr>
<tr>
<td>ACE (Access Control Entry)</td>
<td>An ACL determines if a particular user can perform a particular operation on a specified object. The list contains access control entries (ACE), each of which specifies a user or group and indicates whether certain operations are allowed or denied for that user or group. Using access control provides security for Commander system use. See the Access Control topic for more information.</td>
</tr>
<tr>
<td>ACL (Access Control List)</td>
<td>An ACL determines if a particular user can perform a particular operation on a specified object. The list contains access control entries (ACE), each of which specifies a user or group and indicates whether certain operations are allowed or denied for that user or group. Using access control provides security for Commander system use. See the Access Control topic for more information.</td>
</tr>
<tr>
<td>actual parameter</td>
<td>An actual parameter is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from &quot;formal parameters&quot;: formal parameters define parameters a procedure is expecting, and actual parameters provide values to use at run-time.</td>
</tr>
<tr>
<td>admin</td>
<td>&quot;admin&quot; is a special built-in user that has universal ElectricCommander access. If you log in as admin, you can perform any operation in the system, regardless of access control limitations.</td>
</tr>
<tr>
<td>agent</td>
<td>An agent is an ElectricCommander component that runs on each machine where job steps can execute. The agent works under the Commander server's control to execute job steps, monitor their progress, and record information about their completion. A single agent process can manage multiple job steps executing concurrently on a single machine. See resource.</td>
</tr>
<tr>
<td>artifact</td>
<td>An artifact is a top-level object containing artifact versions, a name template for published artifact versions, artifact specific properties, and access control entries to specify privileges.</td>
</tr>
<tr>
<td>artifact key</td>
<td>An artifact key is an identifier for an artifact and the &quot;key&quot; component of the artifact name.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>artifact repository</td>
<td>See repository.</td>
</tr>
<tr>
<td>artifact version</td>
<td>An artifact version is a collection of 0 to N files that were published to an artifact repository.</td>
</tr>
<tr>
<td>backingstore</td>
<td>The backingstore is the directory on the repository server where artifact versions are stored. By default, the backingstore is the &lt;datadir&gt;/repository-data directory in the repository installation—this default setting can be changed.</td>
</tr>
<tr>
<td>compression</td>
<td>Compression reduces transfer time when publishing an artifact. However, compression also adds overhead when computing the compressed data. If files included in the artifact version are primarily text files or are another highly compressible file format, the benefit of reduced transfer time outweighs the cost of computing compressed data.</td>
</tr>
<tr>
<td>continuous integration</td>
<td>Using continuous integration means a build is launched every time code changes are checked into a Source Control Management (SCM) system. The Commander ElectricSentry component is the engine for continuous integration, while the CI Continuous Integration Dashboard is the front-end user interface for ElectricSentry.</td>
</tr>
<tr>
<td>credential</td>
<td>A credential is an object that stores a user name and password for later use. You can use credentials for user impersonation and saving passwords for use inside steps. Two credential types are available: stored or dynamic.</td>
</tr>
<tr>
<td>custom property</td>
<td>Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner and behave in every way like an intrinsic object-level property with one exception: they are not created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution. Custom properties in a property sheet can be one of two types: string property or a property sheet property. String properties hold simple text values. Property sheet properties hold nested properties. Nested properties are accessed by way of the property sheet property of their containing sheet.</td>
</tr>
</tbody>
</table>
| description          | A description is an optional plain text or HTML description for an object. Description text is for your use, Commander does not use this information. If using HTML, you must surround your text with `<html>` ... `</html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>diagnostic extract</td>
<td>A diagnostic extract is a log file portion from a job step, typically describing an error or interesting condition, extracted by a postprocessor and saved for reporting. The postprocessor usually places this information in an XML file in the top-level job workspace directory, and then sets a property that contains the filename. The Commander postp postprocessor uses filenames like diag-2770.xml, where &quot;2770&quot; is the unique identifier for the step. Other postprocessors you may use can have a different filename configuration.</td>
</tr>
<tr>
<td>dynamic credential</td>
<td>Dynamic credentials are captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded.</td>
</tr>
<tr>
<td>ec-perl</td>
<td>ec-perl is a small wrapper program installed as part of ElectricCommander. When the ec-perl wrapper runs, it sets up the environment, finds, and calls Commander’s copy of Perl, passing all of its parameters to Perl.</td>
</tr>
<tr>
<td>ectool</td>
<td>ectool is the ElectricCommander command-line application that provides control over the Commander system if you prefer using a command-line interface rather than the Commander web interface. Most functions that can be invoked through the Commander web interface can be invoked using ectool.</td>
</tr>
<tr>
<td>ElectricAccelerator</td>
<td>ElectricAccelerator is a software build accelerator that dramatically reduces software build times by distributing the build over a large cluster of inexpensive servers. Using a patented dependency management system, ElectricAccelerator identifies and fixes problems in real time that would break traditional parallel builds. ElectricAccelerator plugs into existing Make-based infrastructures seamlessly and includes web-based management and reporting tools.</td>
</tr>
<tr>
<td>ElectricSentry</td>
<td>ElectricSentry is the ElectricCommander engine for continuous integration—integrating with numerous Source Control Management (SCM) systems. ElectricSentry is installed automatically with Commander and is contained in a Commander plugin named ECSCM and in the Electric Cloud project. Note: The CI Continuous Integration Dashboard is the front-end user interface for ElectricSentry.</td>
</tr>
<tr>
<td>email configuration</td>
<td>Before you can send an email notifier, you must set up and email configuration, which establishes communication between the Commander server and your mail server.</td>
</tr>
<tr>
<td>email notifier</td>
<td>After setting up the Commander server and your mail server to communicate, you can send email notifications (notifiers). You can attach email notifiers to procedures, procedure steps, and state definitions.</td>
</tr>
<tr>
<td>Event log</td>
<td>See log (s).</td>
</tr>
<tr>
<td>Everyone</td>
<td>A special intrinsic access control group that includes all users.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>filter</td>
<td>Two filter categories:&lt;br&gt;  - Intrinsic filters - these filters provide a convenient way to access certain well-defined fields for jobs.&lt;br&gt;    - Custom filters - these filters allow you to access a much broader range of values, including custom properties. Any values accessible through an intrinsic filter can be checked using a custom filter also (though not as conveniently).</td>
</tr>
<tr>
<td>formal parameter</td>
<td>A formal parameter is an object that defines a parameter expected by a procedure, including its name, a default value, and an indication of whether the parameter is required. Formal parameters are different from &quot;actual parameters&quot;: formal parameters define the kinds of parameters a procedure is expecting, and actual parameters provide values to use at run-time.</td>
</tr>
<tr>
<td>gateway</td>
<td>To communicate with a resource, workspace, or artifact repository server in another zone, a &quot;gateway&quot; must be created. A gateway object contains two resource (agent) machines, for example, GatewayResource1 and GatewayResource2—each configured to communicate with the other. One gateway resource resides in the source zone and the other in the target zone. A gateway is bidirectional and informs the Commander server that each gateway machine is configured to communicate with its other gateway machine (in another zone).</td>
</tr>
<tr>
<td>group</td>
<td>A group defines a collection of users for access control purposes. A group can be defined externally in an LDAP or Active Directory repository, or locally in the Commander server. See local group.</td>
</tr>
<tr>
<td>impersonation</td>
<td>Impersonation is a mechanism that allows a job step to execute under a particular login account (the Commander agent &quot;impersonates&quot; a particular user during the execution of that step). Impersonation is implemented using credentials.</td>
</tr>
<tr>
<td>inheritance</td>
<td>A feature of the Commander access control mechanism where access to a particular object is determined by the access control list for that object, and also by the access control lists of the object's parent and other ancestors. Each object can be configured to enable or disable inheritance from its ancestors.</td>
</tr>
<tr>
<td>intrinsic property</td>
<td>Intrinsic properties represent attributes that describe the object to which they are attached. Commander automatically provides intrinsic properties for each similar type object within Commander. For example: Every project has a description property that can be referenced with a non-local property path such as /projects/Examples/description.</td>
</tr>
<tr>
<td>job</td>
<td>A job is the output associated with invoking a Commander procedure. A new job is created each time you run (execute) a procedure.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| job configuration  | A job configuration is an object containing all parameter and credential information needed to run a procedure. A Job Configuration section is provided as part of the Commander Home page to make it easy for you to invoke your favorite configurations with a single mouse click. You can create job configurations in three ways:  
  - From the Job Details page for a previously invoked job, click the **Save Configuration** link at the top of the page. Your saved job configuration will be displayed on your Home page.  
  - Create a job configuration from "scratch" by clicking the **Create** link in the Job Configurations section (on the Home page). In the Create Configuration popup menu, select the project and procedure you want to use for creating this configuration.  
  - On the page for editing a schedule, click the **Save Configuration** link at the top of the page. Your saved configuration will be displayed on your Home page. |
| job name template  | This is the template used to determine the default name for jobs launched from the procedure. You can create a Job Name Template when you create a procedure.  
  For example:  
  In the Job Name Template field, you might supply:  
  `$[projectName]_$/increment /myproject/jobCounter]_[$[timestamp]  
  which produces a name like:  
  `projectFoo_1234_20140102130321`  
  You can supply any combination of elements to create procedure names more meaningful to you. For example, you could choose to include the build number and procedure name. |
| jobs quick view    | A Jobs Quick View section is one of the facilities provided on the Commander Home page. This section allows you to define a category of jobs interesting to you (such as all running jobs or all jobs for a particular product version). Your Home page can display the last several jobs in each category you define. |
| job step           | After a procedure is executed, the resulting job contains one job step for each step in the original procedure. The job step records information about the procedure step execution, such as the command executed, the resource where it executed, execution time, and error information. |
| job workspace      | A directory (containing all files and subdirectories) allocated by Commander for a particular job. Each job workspace is allocated as the child of a workspace root directory.  
  See [workspace](#). |
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>local group</td>
<td>A group defined inside Commander, as opposed to a group defined in an external repository. A local group can refer to both local and remote users, whereas a group in an external repository refers to users in that repository only. See group.</td>
</tr>
<tr>
<td>local user</td>
<td>A user defined inside Commander, as opposed to a user defined in an external repository. If a user defined in an external repository has the same name as a local user, the external user is not accessible. Local users are not visible outside Commander. Electric Cloud recommends using external accounts whenever available, but you may need to create local users if you do not have a shared directory service or if you need special accounts to use for Commander only. See user.</td>
</tr>
<tr>
<td>log(s)</td>
<td>ElectricCommander provides a log for events generated anywhere in the system, including jobs and workflows. <strong>Note:</strong> From the Administration tab, the default view for the Event Log page is the warning (WARN) level. For workflow and job event logs, the default view from their respective pages is the information (INFO) level.</td>
</tr>
<tr>
<td></td>
<td>- To see only events for a single workflow, select the Workflows tab, then a workflow Name to go to the Workflow Details page and click the View Log link at the top of the page.</td>
</tr>
<tr>
<td></td>
<td>- To see only events for a single job, select the Jobs tab, then the Job name to go to the Job Details page and click the View Log link at the top of the page.</td>
</tr>
<tr>
<td></td>
<td>- To see only events for a specific object, select the Search tab to go to the Define Search page.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td>You can select the Object Type, &quot;Log Entry&quot;, then click the <strong>Add Intrinsic Filter</strong> link. Select the down-arrow where you see &quot;Container&quot; auto-populated and select &quot;Container Type. Use the &quot;equals&quot; operator, then select the next down-arrow to choose an object. Click <strong>OK</strong> to start the search.</td>
</tr>
<tr>
<td></td>
<td>See the Event Log topic for more information.</td>
</tr>
<tr>
<td>matcher</td>
<td>A matcher controls the postp postprocessor. Use matchers to extend postp with additional patterns if you find useful patterns in your log files undetected by postp. A matcher contains a pattern that matches lines in a step's log and actions to carry out if/when the pattern matches.</td>
</tr>
<tr>
<td>misfire policy</td>
<td>A misfire policy allows you to manage how a schedule resumes in cases where the normal scheduled time is interrupted. Available options are:</td>
</tr>
<tr>
<td></td>
<td>skip (all misfires are ignored and the job runs at the next scheduled time) and run once (after one or more misfires, the job runs at the soonest time that occurs within an active region).</td>
</tr>
<tr>
<td></td>
<td>See schedule.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>parameter</td>
<td>A property value passed into a procedure when it is invoked (at run time), and used by the procedure to change its behavior. Two types of parameters: actual and formal.</td>
</tr>
<tr>
<td>plugin</td>
<td>A plugin is a collection of one or more features, or a third-party integration or tool that can be added to ElectricCommander. Plugins are delivered as a JAR file containing the functional implementation. When a plugin is installed, the Commander server extracts the JAR contents to disk into a configurable plugins directory. A plugin has an associated project that can contain procedures and properties required by the implementation. A plugin can provide one or more new pages for the web interface and may also provide a configuration page so you can provide additional information that may be necessary to implement the plugin.</td>
</tr>
<tr>
<td>polling frequency</td>
<td>The polling frequency is how often the ElectricSentry continuous integration engine is set to look for new code check-ins. The default is set to every 5 minutes, but this number can be adjusted.</td>
</tr>
<tr>
<td>pool</td>
<td>Also known as &quot;resource pool&quot;. A pool is a collection of resources. If a step specifies a pool name as its resource, Commander can choose any available resource within that pool.</td>
</tr>
<tr>
<td>postp</td>
<td>postp is a postprocessor included with ElectricCommander. postp uses regular expression patterns to detect interesting lines in a step log. postp is already configured with patterns to handle many common cases such as error messages and warnings from gcc, gmake, cl, junit, and cppunit, or any error message containing the string &quot;error.&quot; postp also supports several useful command-line options, and it can be extended using &quot;matchers&quot; to handle environment-specific errors. See matcher.</td>
</tr>
<tr>
<td>postprocessor</td>
<td>A postprocessor is a command associated with a particular procedure step. After a step executes, the postprocessor runs to analyze its results. Typically, a postprocessor scans the step log file to check for errors and warnings. Also, it records useful metrics such as the number of errors in properties on the job step, and extracts step log portions that provide useful information for reporting. Commander includes a standard postprocessor called postp for your use and you can &quot;extend&quot; postp. See matcher.</td>
</tr>
<tr>
<td>preflight build</td>
<td>A preflight build provides a way to build and test a developer's changes before those changes are committed. A &quot;post-commit&quot; source tree is simulated by creating a clean source snapshot and overlaying the developer's changes on top of it. These sources are then passed through the production build procedure to validate the changes work successfully. Developers are allowed to commit their changes only if the preflight build is successful. Because developer changes are built and tested in isolation, many common reasons for broken production builds are eliminated.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>privileges</td>
<td>Commander supports four privilege types (for access control/security) for each object:&lt;br&gt;  • Read - Allows object contents to be viewed.&lt;br&gt;  • Modify - Allows object contents (but not its permissions) to be changed.&lt;br&gt;  • Execute - If an object is a procedure or it contains procedures (for example, a project), this privilege allows object procedures to be invoked as part of a job. For resource objects, this privilege determines who can use this resource in job steps.&lt;br&gt;  • Change Permissions - Allows object permissions to be modified.</td>
</tr>
<tr>
<td>procedure</td>
<td>A procedure defines a process to automate one or more steps. A procedure is the Commander unit you execute (run) to carry out a process. A step in one procedure can call another procedure (in the same or different project), and this procedure then becomes known as a &quot;subprocedure&quot; (also known as a &quot;nested&quot; procedure). The step can pass arguments to the subprocedure.</td>
</tr>
<tr>
<td>project</td>
<td>A project is a top-level container for related procedures, workflows, schedules, jobs, and properties, which is used to isolate different user groups or functions, and also encapsulate shared facilities. Projects have two purposes:&lt;br&gt;  • Projects allow you to create separate work areas for different purposes or groups of people so they do not interfere with each other. In a small organization, you might choose to keep all work within a single project, but in a large organization, you may want to use projects to organize information and simplify management.&lt;br&gt;  • Projects simplify sharing. You can create library projects containing shared procedures and invoke these procedures from other projects. After creating a library project, you can easily copy it to other Commander servers to create uniform processes across your organization.</td>
</tr>
<tr>
<td>project principal</td>
<td><em>Project principal</em> is a special user ID associated with each project. If a project name is &quot;xyz,&quot; the project principal for that project is &quot;project: xyz&quot; (with an embedded space). This principal is used when procedures within the project are run, so you can create access control entries for this principal to control runtime behavior.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| property      | A property is a name-value pair associated with ElectricCommander objects to provide additional information beyond what is already built into the system. Built-in data is accessible through the property mechanism also. Two types of properties: intrinsic and custom. Commander provides Intrinsic properties and allows you to create Custom properties. Note: Intrinsic properties are case-sensitive. Custom properties, like all other object names in the Commander system, are case-preserving, but not case-sensitive.  
- **Intrinsic properties**  
  These properties represent attributes that describe the object to which they are attached, and are automatically by Commander for each similar type object. For example, every project has a Description property that can be referenced with a non-local property path such as /projects/Examples/description.  
- **Custom properties**  
  Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner, and behave in every way like an intrinsic object-level property with one exception: they are not created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution. |
<p>| property sheet | A property sheet is a collection of properties that can be nested to any depth. The property value can be a string or a nested property sheet. Most objects have an associated &quot;property sheet&quot; that contains custom properties created by user scripts. |
| proxy agent    | A proxy agent is an agent on a supported Linux or Windows platform, used to proxy commands to an otherwise unsupported agent platform. Proxy agents have limitations, such as the inability to work with plugins or communicate with ectool commands.?? |
| proxy resource | This resource type requires SSH keys for authentication. You can create proxy resources (agents and targets) for Commander to use on numerous other remote platforms/hosts that exist in your environment. |
| proxy target   | A proxy target is an agent machine on an unsupported platform that can run commands via an SSH server. |
| publisher      | A publisher is the job that completes the publish operation for an artifact version. |</p>
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>quiet time</td>
<td>An inactivity period before starting a build within a continuous integration system. This time period allows developers to make multiple, coordinated check-ins to ensure a build does not start with some of the changes only—assuming all changes are checked-in within the specified inactivity time period. This time period also gives developers an opportunity to &quot;back-out&quot; a change if they realize it is not correct. Using ElectricSentry, the inactivity time period can be configured globally for all projects or individually for a single project.</td>
</tr>
<tr>
<td>reports</td>
<td>ElectricCommander provides multiple reports and custom report capabilities to help you manage your build environment.</td>
</tr>
<tr>
<td></td>
<td>• Real-time reports - filtered view of your workload in real-time</td>
</tr>
<tr>
<td></td>
<td>• Build reports - summary reports produced at the end of a build and attached to the job</td>
</tr>
<tr>
<td></td>
<td>• Batch reports - summaries of your build environment with trends over time, two types:</td>
</tr>
<tr>
<td></td>
<td>• Default Batch reports - automatically installed during ElectricCommander installation and scheduled to run daily (Cross Project Summary, Variant Trend, Daily Summary, Resource Summary, Resource Detail)</td>
</tr>
<tr>
<td></td>
<td>• Optional Batch reports - you can configure, rename, and schedule these reports to fit your requirements (Category Report, Procedure Usage Report, Count Over Time Report, Multiple Series Reports)</td>
</tr>
<tr>
<td></td>
<td>• Custom reports - your choice to create and add at any time</td>
</tr>
<tr>
<td>repository or repository server</td>
<td>The artifact repository is a machine where artifact versions are stored in either uncompressed tar archives or compressed tar-gzip archives. The repository server is configured to store artifact versions in a directory referred to as the repository backingstore. By default, the backingstore is the &lt;datadir&gt;/repository-data directory in the repository installation—this default setting can be changed. A repository is an object that stores artifact versions. This object primarily contains information about how to connect to a particular artifact repository. Similar to steps in a procedure, repository objects are in a user-specified order. When retrieving artifact versions, repositories are queried in this order until one containing the desired artifact version is found. Connection information is stored in the repository object on the Commander server.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| resource  | A resource specifies an agent machine where job steps can be executed. Resources can be grouped into a "pool," also known as a "resource pool." Commander supports two types of resources:  
  - **Standard** - specifies a machine running the ElectricCommander agent on one of the supported agent platforms  
  - **Proxy** - requires SSH keys for authentication. You can create proxy resources (agents and targets) for Commander to use on numerous other remote platforms/hosts that exist in your environment. |
| schedule  | A schedule is an object used to execute procedures automatically in response to system events. For example, a schedule can specify executing a procedure at specific times on specific days. Three types of schedules are available: Standard, Continuous Integration, and Custom (custom schedules are typically continuous integration schedules that do not use the ECSCM plugin). |
| Sentry schedule | A continuous integration schedule created using the ElectricSentry engine for continuous integration or the CI Continuous Integration Dashboard, which is an easy-to-use front-end user interface for the ElectricSentry engine. |
| shortcut  | One type of shortcut is part of the Commander Home page facility and records the location of a page you visit frequently (either inside or outside of ElectricCommander), so you can return to that page with a single click from the Home page.  
Another type of shortcut is a context-relative shortcut to property paths. This shortcut can be used to reference a property without knowing the exact name of the object that contains the property. You might think of a shortcut as another part of the property hierarchy. These shortcuts resolve to the correct property path even though its path elements may have changed because a project or procedure was renamed. Shortcuts are particularly useful if you do not know your exact location in the property hierarchical tree. |
| state     | Workflows always have a single active *state*. Each state in a workflow, when it becomes active, can perform an action. A state can run a procedure to create a subjob or run a workflow definition to create a subworkflow—in the same way that procedures can call other procedures. One or more states can be designated as "starting" states to provide multiple entry points into the workflow.  
See state description. |
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
</table>
| state definition   | Workflow objects are split into two types: *Definition* objects and *Instance* objects. *Definition* objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects.  
When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition.  
**Note:** We omit the "Instance" qualifier for brevity in the API and the UI.  
Each workflow can contain one or more state objects. Defining states for a workflow is analogous to defining steps for a procedure. |
| step               | A step is a procedure component. Each step specifies a command to execute on a particular resource or a subprocedure (nested procedure) to invoke. Commonly created steps include:  
  - **Command** - This step invokes a `bat`, `cmd`, `shell`, `perl` script, or similar.  
  - **Subprocedure** - This step invokes another Commander procedure.  
  - **Plugin step** - These include task-specific steps. Depending on which step-type you choose, the information you need to supply is somewhat different. Some of the step types bundled with ElectricCommander include:  
    - Publish or retrieve artifact version  
    - Send Email  
    - Various SCM step types  
    - Build tools and more |
<p>| stored credential  | <em>Stored credentials</em> are given a name and stored in encrypted form in the database. Each project contains a list of stored credentials it owns. These credentials are managed from the Project Details page.                                                                                                                                                                                                                       |
| subprocedure       | Creating subprocedures is a way of &quot;nesting&quot; procedures. A step (from any procedure) can call a procedure from another project or the same project. The procedure called by the step then becomes a subprocedure.                                                                                                                                                                                                                     |
| substitution       | A mechanism used to include property values in step commands and elsewhere. For example, if a step command is specified as &quot;echo &quot;status&quot;&quot;, and when the step executes there is a property named &quot;status&quot; with value &quot;success&quot;, the actual command executed will be &quot;echo success&quot;.                                                                                                      |
| system object      | This is a special object whose access control lists are used to control access to some ElectricCommander internals. <em>System objects are:</em> <code>admin</code>, <code>artifactVersions</code>, <code>directory</code>, <code>emailConfigs</code>, <code>forceAbort</code>, <code>licensing</code>, <code>log</code>, <code>plugins</code>, <code>priority</code>, <code>projects</code>, <code>repositories</code>, <code>resources</code>, <code>server</code>, <code>session</code>, <strong>and</strong> <code>workspaces</code>.                                                                                              |</p>
<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
</table>
| tag                   | A way to categorize a project to identify its relationship to one or more other projects or groups. You can edit a project to add a tag. Supply a tag if you want to categorize or "mark" a project to identify its relationship to one or more other projects or groups.  

For example, you might want to tag a group of projects as "production" or "workflow", or you might want to use your name so you can quickly sort the project list to see only those projects that are useful to you. |
| transition            | Transitions are used to move workflow progress from one state to another state. Four types of transitions are available to move a workflow to the next state:  

- On Enter - transitions before sending notifiers or starting the sub-action  
- On Start - transitions immediately after starting the sub-action. These transitions are ignored if no sub-action is specified for the source state.  
- On Completion - transitions when the sub-action completes. These transitions are ignored if no sub-action is specified for the source state.  

**Note:** On Completion transitions are taken only if the state is still active when the sub-action completes, and are ignored if the workflow has transitioned to another state—this can occur if an On Start or Manual transition occurred before the sub-action completed.  
- Manual - transitions when a user selects the transition in the UI and specifies parameters. The same action can occur using ectool or the Perl API by calling transitionWorkflow. Only users who have "execute" permission on the transition are allowed to use the Manual transition. See transition definition. |
| transition definition | Workflow objects are split into two types: Definition objects and Instance objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects.  

When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition.  

**Note:** We omit the "Instance" qualifier for brevity in the API and the UI.  

Each state can contain one or more transition objects. The transition definition object requires a name for the transition. This transition name will appear on the Workflow Definition Details page for quick reference and also on the State Definition Details page when you select the Transition Definitions tab.  

You can define one or more transitions for each state, depending on which transition options you want to apply to a particular state. |
| user                  | A user defines an account used to log into the system and control access to ElectricCommander objects. A user can be defined externally in an LDAP or Active Directory repository, or locally in ElectricCommander. See local user. |
You can use a workflow to design and manage processes at a higher level than individual jobs. For example, workflows allow you to combine procedures into processes to create build-test-deploy lifecycles.

A workflow contains *states* and *transitions* you define to provide complete control over your workflow process. The Commander Workflow feature allows you to define an unlimited range of large or small lifecycle combinations to meet your needs. See [workflow definition](#).

Workflow objects are split into two types: *Definition* objects and *Instance* objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects.

When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition.

**Note:** We omit the "Instance" qualifier for brevity in the API and the UI.

This is the template used to determine the default name of jobs launched from the workflow definition. For example:

```
${[projectName]}_${[increment]}/myproject/workflowCounter}_${[timestamp]}
```

(substitute your values for the names above)

Produces a workflow name like:
```
projectName_123_20140102130321
```

A workspace is a subtree of files and directories where job file data is stored. The term "workspace" typically refers to the top-level directory in this subtree.

A workspace root is a directory in which ElectricCommander allocates job workspace directories. Each workspace root has a logical name used to refer to it in steps and procedures.

A zone is a way to partition a collection of agents to secure them from use by other groups—similar to creating multiple top-level networks. For example, you might choose to create a developers zone, a production zone, and a test zone—agents in one zone cannot directly communicate with agents in another zone.

A default zone is created during Commander installation. The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).