ElectricCommander 5.1
API Guide

Electric Cloud, Inc.
www.electric-cloud.com

Document Rev. 1

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

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

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 proc1 --actualParameter parm1=value1 p
arm2=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 => 'proc1',
                    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:

```
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:

```
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.

```
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:

```
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

```
$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:

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

or

```
$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 ElectricFlowreturned XML block. See documentation on CPAN for more information.

```
$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:

```
$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:

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

Specifying a named object

Any API argument that refers to a named object (for example, `projectName`, `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:

```
$cmdr->getProject ('$[/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.

Global Arguments	Description
`--help`	Display an online version of ectool commands with a short description. Displays command information if followed by a command name.
`--version`	Display the ectool version number.
`--server <hostname>`	ElectricCommander server address. Defaults to the COMMANDER_SERVER environment variable. If this variable does not exist, the default is to the last server contacted through the API. However, if there is no record for which server was contacted, 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.
`--port <port>`	HTTP listener port on the ElectricCommander server. Defaults to port 8000.
`--securePort <secureport>`	HTTPS listener port on the ElectricCommander server. Defaults to port 8443.
`--secure <0|1>`	Use HTTPS to communicate with the Commander server. **Note:** Certain requests (for example, `login`, `createUser`, and `modifyUser`) automatically use HTTPS because passwords are being sent, which means it is not necessary to specify `secure` for those APIs. Defaults to `1`.

Global Arguments	Description
`--timeout <s>`	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.
`--retryTimeout <s>`	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.
`--retry <0|1>`	Retry the request if it times out based on the "timeout" value. Default is "0" and should rarely be changed.
`--user <`*`username`*`>`	Use the session associated with the user. Defaults to the user who last logged in.
`--service <`*`spn`*`>`	Specify the service principal name to use for Kerberos. Defaults to HTTP@host.domain.
`--setDefault <0|1>`	Use the current session as the default for subsequent invocations. Defaults to `1`.
`encoding <`*`charEncoding`*`>`	Use the specified encoding for input/output. For example, for `charEncoding`, supply UTF-8, cp 437, and so on. Default is autodetected.
`--dryrun`	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.
`--silent`	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.
`--valueOf`	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 <path>` 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 "4.1.0.48418".
`--format <`*`format`*`>`	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`
`--ignoreEnvironment`	Force ectool to ignore COMMANDER_ENV variables.

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>
```

```
        <propertyId>199827</propertyId>
        <propertyName>p1</propertyName>
        <createTime>2010-07-21T16:41:20.003Z</createTime>
        <expandable>1</expandable>
        <lastModifiedBy>project: EA Articles</lastModifiedBy>
        <modifyTime>2010-07-21T16:41:20.003Z</modifyTime>
        <owner>project: EA Articles</owner>
        <value>99</value>
      </property>
    </response>
    <response requestId="2">
      <property>
        <propertyId>199828</propertyId>
        <propertyName>p2</propertyName>
        <createTime>2010-07-21T16:41:20.019Z</createTime>
        <expandable>1</expandable>
        <lastModifiedBy>project: EA Articles</lastModifiedBy>
        <modifyTime>2010-07-21T16:41:20.019Z</modifyTime>
        <owner>project: EA Articles</owner>
        <value>1</value>
      </property>
    </response>
  </responses>
```

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:

```
# 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:

   ```
   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:

   ```
   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:

  ```
  #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\li
b',
  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

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.

Arguments	Descriptions
artifactName	The name of the artifact.
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. A qualifying project name is **required**. **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 full name of the group. For Active Directory and LDAP, the full name if the full DN.

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.
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. This value is a "handle" only for passing to API commands. The internal structure of this value is subject to change - do not parse this value.
pluginName	The plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.
procedureName	The name of the procedure or a path to a procedure, including the name. **Also requires** projectName
projectName	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.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
repositoryName	The name of the repository used for artifact management.
resourceName	The name of a resource.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of a schedule - may be a path to a schedule. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step - may be a path to the step. **Also requires** projectName and procedureName
systemObjectName	System objects names include: admin|artifactVersions|directory|emailConfigs|log|plugins| server|session|workspaces
transitionDefinitionName	The name of the transition definition.

Arguments	Descriptions
transitionName	The name of the transition.
userName	The full name of a user (for Active Directory or LDAP, this may be user@domain).
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of a workspace.
zoneName	The name of the zone.

Positional arguments

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

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->breakAclInheritance ({ projectName => "Sample Project"});
```

ectool

syntax: `ectool breakAclInheritance ...`

Example

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

Back to Top

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.

Arguments	Descriptions
applicationName	The name of the application container of the property sheet which owns the property; must be unique among all projects.

Arguments	Descriptions
applicationTierName	The name of the application tier container of the property sheet which owns the property.
artifactName	The name of the artifact container of the property sheet which owns the property.
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. 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.
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.

Arguments	Descriptions
notifierName	The name of the email notifier.
objectId	This is an object identifier returned by findObjects and getObjects. This value is a "handle" only for passing to API commands. The internal structure of this value is subject to change - do not parse this value.
path	Property path string.
pluginName	The name of the plugin - the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.
procedureName	The name of the procedure - may be a path to the procedure. **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 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.
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 the resource.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule - may be a path to the schedule. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step - may be a path to the step. **Also requires** projectName and procedureName
systemObjectName	System object names include: admin|directory|licensing|log|plugins|priority|projects|

Arguments	Descriptions
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
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.
workflowName	The name of the workflow.
workspaceName	The name of the workspace.
zoneName	The name of the zone.

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"
```

Back to Top

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	Descriptions
artifactName	The name of the artifact.

Arguments	Descriptions
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.
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: readPrivilege modifyPrivilege executePrivilege changePermissionsPrivilege	<allow|deny> If a privilege is not specified, permission is set to inherit from its parent object's ACL.
procedureName	The name of the procedure. **Also requires** projectName

Arguments	Descriptions
projectName	The name of the project.
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 the resource.
resourcePoolName	The name of a pool containing one or more resources.
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 object names include: admin|artifacts|directory|emailConfigs|forceAbort|licensing| log|plugins|priority|projects|repositories|resources|server| session|test|workspaces|zonesAndGateways
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

None or status OK message.

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
```

Back to Top

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.

Arguments	Descriptions
`artifactName`	The name of the artifact.
`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.
`configName`	The name of the email configuration.
`gatewayName`	The name of the gateway.

Arguments	Descriptions
groupName	The name of a group whose ACL entry you want to delete.
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.
objectId	An object identifier returned by findObjects and getObjects.
notifierName	The name of the email notifier whose ACL entry you want to delete.
pluginName	The name of the plugin whose ACL entry you want to delete.
principalName	This is either the user or the group name.
principalType	This is either a user or a group <user|group>. Defaults to "user".
procedureName	The name of the procedure whose ACL entry you want to delete. **Also requires** projectName, where this procedure is a member.
projectName	The name of the project where you are deleting an ACL entry.
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 the resource whose ACL entry you want to delete.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule whose ACL entry you want to delete. **Also requires** projectName from which this schedule runs procedures.
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step whose ACL entry you want to delete. **Also requires** projectName and procedureName to indicate where this step resides.

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"
```

Back to Top

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.

Arguments	Descriptions
applicationName	The name of the application container of the property sheet which owns the property; must be unique among all projects.
applicationTierName	The name of the application tier container of the property sheet which owns the property.
artifactName	The name of the credential container of the property sheet which owns the property.The name of the artifact.
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.
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.
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.

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.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier that contains the ACL.
objectId	This is an object identifier returned by `findObjects` and `getObjects`.
path	Property path string.
pluginName	The name of the plugin that contains the ACL.
procedureName	The name of the procedure containing the ACL. **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 ACL; must be unique among all projects.
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 the resource that contains the ACL.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the ACL. **Also requires** `projectName`
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the ACL. **Also requires** `projectName` and `procedureName`

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

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

Example

```
$cmdr->getAccess({projectName => "Sample Project"});
```

ectool

syntax: `ectool getAccess ...`

Example

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

Back to Top

getAclEntry

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.

Arguments	Descriptions
`artifactName`	The name of the artifact.
`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.
`configName`	The name of the email configuration.
`gatewayName`	The name of the gateway.
`groupName`	The name of the 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 the user or group name.
`principalType`	This is either `user` or `group`.
`procedureName`	The name of the procedure. **Also requires** `projectName`
`projectName`	The name of the project.

Arguments	Descriptions
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 the resource.
resourcePoolName	The name of a pool containing one or more resources.
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

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

ectool

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

Example

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

Back to Top

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.

Arguments	Descriptions
`artifactName`	The name of the artifact.
`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.
`configName`	The name of the email configuration.
`gatewayName`	The name of the gateway.
`groupName`	The name of the group containing the ACL entry.
`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.

Arguments	Descriptions
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of the email notifier containing the ACL entry.
objectId	This is an object identifier returned by findObjects and getObjects.
pluginName	The name of the plugin containing the ACL entry.
Privileges: readPrivilege modifyPrivilege executePrivilege changePermissionsPrivilege	<allow|deny>
principalName	This is either the user or group name.
principalType	This is either user or group.
procedureName	The name of the procedure containing the ACL entry. **Also requires** projectName
projectName	The name of the project containing the ACL entry.
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 the resource containing the ACL entry.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the ACL entry. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the ACL entry. **Also requires** projectName and procedureName
systemObjectName	System object names include: admin|artifacts|directory|emailConfigs|forceAbort| licensing|log|plugins|priority|projects| repositories|resources|server|session|test| workspaces|zonesAndGateways

Arguments	Descriptions
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user containing the ACL entry.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing the ACL entry.
zoneName	The name of the zone.

Positional arguments

```
principalType, principalName
```

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

Back to Top

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.

Arguments	Descriptions
artifactName	The name of the artifact.

Arguments	Descriptions
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	Descriptions
projectName	The name of the project whose ACL inheritance you want to restore.
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 the resource whose ACL inheritance you want to restore.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule whose ACL inheritance you want to restore. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step whose ACL inheritance you want to restore. **Also requires** projectName and procedureName
systemObjectName	The name of the system object whose ACL inheritance you want to restore. 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 whose ACL inheritance you want to restore.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace whose ACL inheritance you want to restore.
zoneName	The name of the zone.

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

API commands - Applications

createApplication

Creates a new application for a project.

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

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String
`applicationName`	Name of the application; must be unique among all projects. Argument Type: String
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricCommander. Argument Type: String

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.

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String
`applicationName`	Name of the application; must be unique among all projects. Argument Type: String

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
```

Back to Top

getApplication

Finds an application by name.

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

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String
`applicationName`	Name of the application; must be unique among all projects. Argument Type: String

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
```

Back to Top

getApplications

Retrieves all applications in a project.

You must specify the `projectName` argument.

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String

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.

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

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
```

Back to Top

API commands - Application Tier

createApplicationTier

Creates a new application tier in the application.

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

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String
`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
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricCommander. Argument Type: String

Response

Returns an application tier element.

ec-perl

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

Example

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

ectool

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

Example

```
ectool createApplicationTier default newApp appTier1
 --description example_text
```

Back to Top

deleteApplicationTier

Deletes a tier from an application.

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

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String
`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

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
```

Back to Top

getApplicationTier

Finds an application tier by name.

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

Arguments	Descriptions
projectName	Name for the project; must be unique among all projects. Argument Type: String
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 an application tier element.

ec-perl

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

Example

```
$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.

Arguments	Descriptions
projectName	Name for the project; must be unique among all projects. Argument Type: String
applicationName	Name of the application; must be unique among all projects. Argument Type: String

Response

Retrieves zero or more application tier elements.

ec-perl

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

Example

```
$ec->getApplicationTiers("Default", "app1");
```

ectool

syntax:`ectool getApplicationTiers <projectName> <applicationName>`

Example

```
ectool getApplicationTiers default newApp
```

Back to Top

getApplicationTiersInComponent

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

You must specify the `projectName` and the `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
`applicationName`	(Optional) Name of an application to which this component is scoped. Argument Type: String

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...]`

Example

```
ectool getApplicationTiersInComponent default newComponent
```

Back to Top

modifyApplicationTier

Modifies an existing tier in the application.

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

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String
`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
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricCommander. Argument Type: String
`newName`	New name for an existing object that is being renamed. Argument Type: String

Response

Retrieves an updated application tier element.

ec-perl

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

Example

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

ectool

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

Example

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

Back to Top

API commands - Artifact Management

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`.

Arguments	Descriptions
`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.
`dependentArtifactVersions`	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: `<groupId>:<artifactKey>:<versionRange>` (`versionRange` is optional). The version range syntax is standard number interval notation. `()` marks exclusive ranges and `[]` marks inclusive ranges.

Positional arguments

`artifactVersionName`

Response

None or status OK message.

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 tha
t
# 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,]"]});
```

ectool

syntax: `ectool addDependentsToArtifactVersion <artifactVersionName>...`

Example

```
ectool addDependentsToArtifactVersion --artifactVersionName "myGroup:myAKey:1.0.0-5
5",
   --dependentArtifactVersions "cmdr:SDK:1.2.0" "core:infra:[2.1.0,]"
```

Back to Top

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`.

Arguments	Descriptions
`cacheDirectory`	The directory where stale artifact versions are stored.
`force`	<*Boolean flag* - `0|1|true|false`> If set to "true", this option can be used so you can cleanup the artifact cache if you are not logged in as "admin".

Positional arguments

`cacheDirectory`

Response

Returns a list of directories that were deleted.

ec-perl

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

Example

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

ectool

syntax: `ectool cleanupArtifactCache <cacheDirectory>`

Example

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

Back to Top

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`.

Arguments	Descriptions
`backingStoreDirectory`	The repository directory where artifact versions are stored.
`force`	<*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

```
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

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

Back to Top

createArtifact

Creates a new artifact.

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

Arguments	Descriptions
`artifactKey`	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
`artifactVersionNameTemplate`	A template for the names of artifact versions published to this artifact. This option overrides the value set in the server settings for "artifact name template.". The global setting can be manipulated in the Server Settings page (Administration > Server, select the Settings link).
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`groupId`	A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, spaces, underscores, hyphens, and periods.

Positional arguments

`groupId, artifactKey`

Response

Returns an `artifact` element.

ec-perl

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

Example

```
$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"
```

Back to Top

createRepository

Creates a repository for one or more artifacts.

You must specify a `repositoryName`.

Arguments	Descriptions
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`repositoryDisabled`	<*Boolean flag* -`0|1|true|false`> Determines whether the repository is disabled. Default is "false".
`repositoryName`	The name of the artifact repository.
`url`	The URL to use to communicate with the repository server.
`zoneName`	The name of the zone where this repository resides.

Positional arguments

`repositoryName`

Response

Returns a `repository` element.

ec-perl

syntax: `$cmdr->createRepository(<repositoryName>, {<optionals>});`

Example

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

ectool

syntax: `ectool createRepository <repositoryName> ...`

Example

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

Back to Top

deleteArtifact

Deletes an existing artifact element and all artifact versions.

You must specify an `artifactName`.

Arguments	Descriptions
`artifactName`	The name of the artifact to delete.

Positional arguments

`artifactName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteArtifact(<artifactName>);`

Example

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

ectool

syntax: `ectool deleteArtifact <artifactName>`

Example

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

Back to Top

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`.

Arguments	Descriptions
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.

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`.

Arguments	Descriptions
repositoryName	The name of the artifact repository to delete.

Positional arguments

repositoryName

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteRepository ("cmdrReposOne");
```

ectool

syntax: `ectool deleteRepository <repositoryName>`

Example

```
ectool deleteRepository cmdrReposOne
```

Back to Top

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 abo
ve.
$cmdr->findArtifactVersions({groupId => "core",
                         artifactKey => "infra",
                              filter => {propertyName => "qaLevel",
```

```
                                operator => "greaterOrEqual",
                                operand1 => "3"}});
```

ectool

Not supported.

Back to Top

getArtifact

Retrieves an artifact by name.

You must specify an `artifactName`.

Arguments	Descriptions
`artifactName`	The name of the artifact.

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"
```

Back to Top

getArtifacts

Retrieves all artifacts in the system.

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

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more `artifact` elements.

ec-perl

syntax: `$cmdr->getArtifacts ();`

Example

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

ectool

syntax: `ectool getArtifacts`

Example

```
ectool getArtifacts
```

Back to Top

getArtifactVersion

Retrieves an artifact version by its name.

You must specify an `artifactVersionName`.

Arguments	Descriptions
`artifactVersionName`	The name of the artifact version to retrieve. **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.
`includeRetrieverJobs`	<*Boolean flag -* `0|1|true|false`> If set to 1, this argument includes `jobId` and `jobName` in returned information. A retriever job is any job that has retrieved the artifact version.
`includeRetrieverJobSteps`	<*Boolean flag -* `0|1|true|false`> If set to 1, this argument includes `jobId`, `jobName`, and `jobStepId` information. A retriever job is any job that has retrieved the artifact version. Because there is no bound to how many job steps may retrieve a given artifact version, the server limits the response to the most recent 200 job steps.
`maxRetrievers`	If one of the `includeRetriever`* options are specified, return at most "this many" of the most recent retrievers. Without this option, the Commander server will return all retrievers.

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

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

ectool

syntax: `ectool getArtifactVersion <artifactVersionName> ...`

Example

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

Back to Top

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.

Arguments	Descriptions
`artifactName`	The name of the artifact for the versions to retrieve.
`retrieverJobId`	The job ID that retrieved an artifact.
`retrieverJobStepId`	The job step ID that retrieved an artifact.

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`.

Arguments	Descriptions
`artifactVersionName`	The name of the artifact version whose manifest you want to retrieve.

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.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more `repository` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getRepositories`

Example

```
ectool getRepositories
```

Back to Top

getRepository

Retrieves an artifact repository by its name.

You must specify a `repositoryName`.

Arguments	Descriptions
repositoryName	The name of the artifact repository to retrieve.

Positional arguments

`repositoryName`

Response

One `repository` element.

ec-perl

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

Example

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

ectool

syntax: `ectool getRepository <repositoryName>`

Example

```
ectool getRepository myRepository
```

Back to Top

modifyArtifact

Modifies an existing artifact.

You must specify an `artifactName`.

Arguments	Descriptions
`artifactName`	The name of the artifact to modify.
`artifactVersionNameTemplate`	A template for the names of artifact versions published to this artifact. This option overrides the value set in the server settings for "artifact name template." The global setting can be manipulated in the Server Settings page (Administration > Server, select the Settings link).
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`

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"});
```

ectool

syntax: `ectool modifyArtifact <artifactName> ...`

Example

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

Back to Top

modifyArtifactVersion

Modifies an existing artifact version.

You must specify an `artifactVersionName`.

Arguments	Descriptions
`artifactVersionName`	The name of the artifact version to modify. **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.
`artifactVersionState`	The state of the artifact version. <`publishing|available|unavailable`>.
`dependentArtifactVersions`	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: `<groupId>:<artifactKey>:<versionRange>` (version range is optional). **Note:** The absence of this argument does not clear or modify the dependent artifact version list for this artifact version.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`newName`	Any name you choose to use as the new name for this artifact version.
`removeAllDependentArtifactVersions`	<*Boolean flag* - `0|1|true|false`> Defaults to "false." Removes all dependent artifacts from this artifact version. Subsequent "retrieves" will no longer retrieve dependent artifacts for this artifact version.
`repositoryName`	The name of the artifact repository.

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 una
vailable
```

Back to Top

modifyRepository

Modifies an existing artifact repository.

You must specify a `repositoryName`.

Arguments	Descriptions
`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>   <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 repository.
`repositoryDisabled`	<*Boolean flag* - `0|1|true|false`> Marks the repository as enabled or disabled. If you do not supply this option, the state of the repository is unchanged.
`repositoryName`	The name of the artifact repository.
`url`	The URL used to communicate with the artifact repository.
`zoneName`	The name of the zone where this repository resides.

Positional arguments

`repositoryName`

Response

Returns a modified `repository` element.

ec-perl

syntax: `$cmdr->modifyRepository (<repositoryName>, {<optionals>});`

Example

```
$cmdr->modifyRepository("myNewRepos", {newName => "cmdrRepository"});
```

ectool

syntax: `ectool modifyRepository <repositoryName> ...`

Example

```
ectool modifyRepository myNewRepos --newName cmdrRepository
```

Back to Top

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`.

Arguments	Descriptions
`repositoryName`	The name of the artifact repository you need to move.
`beforeRepositoryName`	Moves this repository (`repositoryName`) to a place before the name specified by this option. If omitted `repositoryName` is moved to the end.

Positional arguments

`repositoryName`

Response

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

ec-perl

syntax: `$cmdr->moveRepository(<repositoryName>, {<optionals>});`

Example

```
$cmdr->moveRepository(reposThree, {beforeRepositoryName => "reposOne"});
```

ectool

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

Example

```
ectool moveRepository reposThree --beforeRepositoryName reposOne
```

Back to Top

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`.

Arguments	Descriptions
`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.
`compress`	*<Boolean flag -* `0|1|true|false`> Default is "true". Controls whether or not the artifact version is compressed during transport, which improves performance for cases where artifact version files are compressible, saving network bandwidth. Where artifact version files are not compressible, performance is reduced. Another consideration is that the artifact version is stored compressed/uncompressed based on this setting in the repository backing-store.
`dependentArtifactVersions`	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: `<groupId>:<artifactKey>:<versionRange>` (`versionRange` is optional). The version range syntax is standard number interval notation. `()` marks exclusive ranges and `[]` marks inclusive ranges.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`excludePatterns`	Semi-colon delimited list of file-path patterns indicating which files/directories under "`fromDirectory`" to exclude when publishing an artifact version. Defaults to "empty," which means no files are excluded. See more information on "pattern syntax" below.

Arguments	Descriptions
`followSymlinks`	*<Boolean flag -* `0|1|true|false`> Default is "true". If true, follow symbolic links and record the target file contents with the symbolic link name in the artifact. If false, record the symbolic link as a symbolic link. Following symbolic links causes the publish API to remain compatible with previous releases.
`fromDirectory`	The directory containing files to publish as the artifact version. A subset of files can be published based on `includePatterns` and `excludePatterns`.
`groupId`	A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
`includePatterns`	Semi-colon delimited list of file-path patterns indicating which files/directories under "`fromDirectory`" to publish in the artifact version. Defaults to "empty," which means all files will be included. Conversely, if only two files are "included," no other files except those two will be included. See more information on "pattern syntax" below.
`repositoryName`	The name of the artifact repository where you want to publish.
`version`	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: `\:<>|?*/` If a version argument is provided, but does not follow the above format, the version will be considered `0.0.0-<user-specified-version-arg>-0` implicitly. See examples below.

Version number examples

User Input	Interpretation		
	Major.Minor.Patch	Qualifier	Build Number
`1`	1.0.0		0
`1.0`	1.0.0		0
`1.0-frank`	1.0.0	`frank`	0
`1.0-36`	1.0.0		36
`1.0-frank-36`	1.0.0	`frank`	36

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"
   --dependentArtifactVersion "cmdr:SDK:1.2.0":"core:infra"
```

Back to Top

removeDependentsFromArtifactVersion

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

You must specify the `artifactVersionName`.

Arguments	Descriptions
artifactVersionName	The name of the artifact version from which you want to remove dependents. **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.
dependentArtifactVersions	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: <groupId>:<artifactKey>:<versionRange> (versionRange is optional). The version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.

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"
```

Back to Top

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.

Arguments	Descriptions
`artifactKey`	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
`artifactName`	The name of the artifact.
`artifactVersionName`	The name of the artifact version.
`cacheDirectory`	The directory where the artifact version is stored. **Note:** The artifact version files are stored in a subdirectory under this cache directory.

Arguments	Descriptions

filters	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" 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	Descriptions
	`or (2 or more operands)`
`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.
`overwrite`	Options are: • `true` – deletes previous content in the directory and replaces the content with your new version. • `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. • `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.
`repositoryNames`	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.
`toDirectory`	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. • If the artifact version is in a local cache directory. it will be copied out of the cache. • 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.
`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

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.

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String
`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
`componentName`	Name of the component. Argument Type: String
`componentProjectName`	(Optional) Project name of the component. Argument Type: String

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
```

Back to Top

createComponent

Creates a new component for a project.

You must specify the `projectName`, `applicationName`, and `applicationTierName` 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
`pluginName`	Name of the plugin. Argument Type: String
`applicationName`	(Optional) Name of an application to scope this component to. Argument Type: String
`credentialName`	(Optional) Name of a credential to attach to this component. Argument Type: String
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricCommander. Argument Type: String

Response

Returns a version control component element.

ec-perl

Syntax:

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

Example:

```
$ec->createComponent("default", "component1", "Publish Artifact Version",
 {description => "New agent"});
```

ectool

Syntax:

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

Example:

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

Back to Top

deleteComponent

Deletes a 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
`applicationName`	(Optional) The name of an application to which this component is scoped. Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

```
$ec->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.

Arguments	Descriptions
`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
```

Back to Top

getComponents

Retrieves all components in a project.

You must specify the `projectName` argument.

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String
`applicationName`	(Optional) Name of the application. Specify to search for components scoped to an application. Argument Type: String

Response

Retrieves zero or more component elements.

ec-perl

Syntax:

```
$<object>->getComponents(<projectName>, {<optionals>);
```

Example:

```
$ec->getComponents("default");
```

ectool

Syntax:

```
ectool getComponents <projectName> [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.

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument Type: String

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:

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

Example:

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

ectool

Syntax:

```
ectool getComponentsInApplicationTier <projectName> <applicationName>
 <applicationTierName>
```

Example:

```
ectool getComponentsInApplicationTier default newApp appTier1
```

Back to Top

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	Descriptions
credential Name	(Optional) Name of the credential. Argument Type: String
description	(Optional) Comment text describing this component; not interpreted at all by ElectricFlow. Argument Type: String
newName	(Optional) New name of the component. Argument Type: String

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
```

Back to Top

removeComponentFromApplicationTier

Removes the given component from the given application tier.

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

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

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
componentName	Name of component. Argument Type: String

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

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.

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.
`procedureName`	The name of a procedure within the "named" project where this credential will be attached.
`projectName`	The name of the project that contains the object where this credential will be attached.
`scheduleName`	The schedule name for running one of the procedures within the "named" project.
`stepName`	A step name within one of the procedures contained in the "named" project.

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"
```

Back to Top

createCredential

Creates a new credential for a project.

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

Arguments	Descriptions
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`credentialName`	The name of the credential to create (any name you choose).
`password`	The password matching the specified user name.
`projectName`	The name of the project where the credential will be stored.
`userName`	The name of the user.

Positional arguments

`projectName, credentialName`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool createCredential <projectName> <credentialName> --userName <userName> --password <password> ...`

Example

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

Back to Top

deleteCredential

Deletes a credential.

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

Arguments	Descriptions
`projectName`	The name of the project that contains this credential.
`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.

Positional arguments

`projectName, credentialName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteCredential <projectName> <credentialName>`

Example

```
ectool deleteCredential "Sample Project" "Build User"
```

Back to Top

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`).

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.
`procedureName`	The name of the procedure that contains the step with the credential to detach.
`projectName`	The name of the project that contains this credential, schedule, or procedure step.
`scheduleName`	The name of the schedule where this credential may be attached.
`stepName`	The name of the step where this credential may be attached. **Also requires** `procedureName`.

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

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

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

Back to Top

getCredential

Finds a credential by name.

You must specify `projectName` and `credentialName`.

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.
`projectName`	The name of the project containing this credential.

Positional arguments

`projectName, credentialName`

Response

One `credential` element.

ec-perl

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

Example

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

ectool

syntax: `ectool getCredential <projectName> <credentialName>`

Example

```
ectool getCredential "Sample Project" "Build User"
```

Back to Top

getCredentials

Retrieves all credentials in a project.

You must specify a `projectName`.

Arguments	Descriptions
`projectName`	The name of the project containing these credentials.
`usableOnly`	<*Boolean flag* - `0|1|true|false`> If set to 1, only those credentials that the currently logged-in user has execute privileges for will be returned.

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
```

Back to Top

getFullCredential

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

You must specify the `credentialName`.

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.
value	<password|userName> If specified, returns only the password or user name.

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

Example

```
# Returns an xPath object containing the password.
my $xpath = $cmdr->getFullCredential("myCred", {value => "password"});


# Parse password from response.
my $password = $xpath->find("//password");
```

ectool

syntax: ectool getFullCredential <credentialName> ...

Example

```
ectool getFullCredential myCred --value password
```

Back to Top

modifyCredential

Modifies an existing credential.

You must specify projectName and credentialName.

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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
newName	Supply any name of your choice to rename the credential.
password	The password for the specified user name.
projectName	The name of the project containing this credential.
userName	The name of the user containing this credential.

Positional arguments

projectName, credentialName

Response

None or a status OK message.

ec-perl

syntax: $cmdr->modifyCredential(<projectName>, <credentialName>, {<optionals>});

Example

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

ectool

syntax: ectool modifyCredential <projectName> <credentialName> ...

Example

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

Back to Top

API Commands - Database Configuration

getDatabaseConfiguration

Retrieves the current database configuration.

Arguments	Descriptions
None	

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: `$cmdr->getDatabaseConfiguration();`

Example

```
$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:

```
customDatabaseDialect
customDatabaseDriver
customDatabaseUrl
```

Arguments	Descriptions
customDatabaseDialect	Class name of the Hibernate dialect *(advanced use only*--the server will choose an appropriate dialect based on the databaseType).
customDatabaseDriver	Class name of the JDBC driver (*advanced use only*--the server will choose an appropriate driver based on the databaseType).
customDatabaseUrl	The JDBC to use (*advanced use only*--the server will compose an appropriate URL).
databaseName	The name of the database you want the Commander server to use.
databaseType	The type of database you want the Commander server to use. Supported database types are: <builtin|mysql|sqlserver|oracle>
hostName	The name of the host machine where the database is running.
ignorePasskeyMismatch	<*Boolean flag* - 0|1|true|false> If the server is started with a different passkey, ignore the mismatch if "true". **Note:** This action discards all saved passwords.
ignoreServerMismatch	<*Boolean flag* - 0|1|true|false> If the server is started on a different host than where the server previously started, ignore the mismatch if "true".
password	The password required to access the database. setDatabaseConfiguration does not allow a passwordless database user. Make sure the database user has a password.
port	The port number used to access the database.
preserveSessions	<*Boolean flag* - 0|1|true|false> If ignoring a server mismatch, default behavior invalidates all sessions. Setting this flag to "true" preserves all sessions, allowing the server to reconnect to running jobs. This option is used in combination with ignoreServerMismatch.
userName	The name of the user required to access the database.

Positional arguments

None

Response

None or a status OK message.

ec-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

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

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

Creates a new Active Directory or LDAP directory provider.

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

Arguments	Descriptions
`commonGroupNameAttribute`	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.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`domainName`	The domain name from which Active Directory server(s) are automatically discovered.
`emailAttribute`	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.
`enableGroups`	*<Boolean flag* -`0|1|true|false`> Determines whether or not to enable external groups for the directory provider. Defaults to "true".
`fullUserNameAttribute`	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.
`groupBase`	This string is prepended to the `basedn` to construct the directory DN that contains group records.

Arguments	Descriptions
groupMemberAttributes	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.
groupMemberFilter	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: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
groupNameAttribute	The group record attribute that contains the name of the group.
groupSearchFilter	This LDAP query is performed in the context of the groups directory to enumerate group records.
managerDn	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. **Note:** This user does not need to be an admin user with modify privileges.
managerPassword	If the managerDn property is set, this password is used to authenticate the manager user.
providerName	This human-readable name will be displayed in the user interface to identify users and groups that come from this provider.
providerType	<ldap|activedirectory>
realm	This is an identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in "same name" collisions across multiple directory providers. The realm is appended to the user or group name when stored in the Commander server. For example, <*user*>@dir (where the realm is set to "dir").
url	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 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).

Arguments	Descriptions
`userBase`	This string is prepended to the `basedn` to construct the directory DN that contains user records.
`userNameAttribute`	The attribute in a user record that contains the user's account name.
`userSearchFilter`	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

`providerName, providerType, url`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

Back to Top

deleteDirectoryProvider

Deletes an Active Directory or LDAP directory provider.

You must specify a `providerName`.

Arguments	Descriptions
providerName	The name of the directory provider you want to delete.

Positional arguments

providerName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteDirectoryProvider <providerName>`

Example

```
ectool deleteDirectoryProvider AD3
```

Back to Top

getDirectoryProvider

Retrieves a directory provider by name.

You must specify a `providerName`.

Arguments	Descriptions
providerName	The name of the directory provider.

Positional arguments

providerName

Response

One `directoryProvider` element.

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

ec-perl

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

Example

```
$cmdr->getDirectoryProvider("AD3");
```

ectool

syntax: `ectool getDirectoryProvider <providerName>`

Example

```
ectool getDirectoryProvider AD3
```

Back to Top

getDirectoryProviders

Retrieves all directory providers.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more `directoryProvider` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getDirectoryProviders`

Example

```
ectool getDirectoryProviders
```

Back to Top

modifyDirectoryProvider

Modifies an existing LDAP directory provider.

You must specify the `providerName`.

Arguments	Descriptions
commonGroupNameAttribute	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.
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
domainName	The domain from which Active Directory servers are automatically discovered.
emailAttribute	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.
enableGroups	<*Boolean flag* - 0|1|true|false> Determines whether or not to enable external groups for the directory provider. Defaults to "true".
fullUserNameAttribute	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.
groupBase	This string is prepended to the basedn to construct the directory DN that contains group records.
groupMemberAttributes	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.
groupMemberFilter	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: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
groupNameAttribute	The group record attribute that contains the name of the group.
groupSearchFilter	A filter name: this LDAP query is performed in the context of the groups directory to enumerate group records.

Arguments	Descriptions
managerDn	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. **Note:** This user does not need to be an admin user with modify privileges.
managerPassword	If the managerDn property is set, this password is used to authenticate the manager user.
newName	Supply any name of your choice to rename the directory provider.
providerName	This human readable name will be displayed in the user interface to identify users and groups that come from this provider.
providerType	<ldap|activedirectory>
realm	This is an identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in "same name" collisions across multiple directory providers. The realm is appended to the user or group name when stored in the Commander server. For example, <*user*>@dir (where the realm is set to "dir").
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).
userBase	This string is prepended to the basedn to construct the directory DN that contains user records.
userNameAttribute	The attribute in a user record that contains the user's account name.
userSearchFilter	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.

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
```

Back to Top

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`.

Arguments	Descriptions
providerName	The name of the directory provider to move.
beforeProviderName	Moves this directory provider (`providerName`) to a place before the name specified by this option. If omitted, `providerName` is moved to the end.

Positional arguments

providerName

Response

None or a status OK message.

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).

Arguments	Descriptions
`commonGroupNameAttribute`	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.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`domainName`	The domain from which Active Directory servers are automatically discovered.
`emailAttribute`	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.
`enableGroups`	<*Boolean flag* - `0|1|true|false`> Determines whether or not to enable external groups for the directory provider. Defaults to "true".
`fullUserNameAttribute`	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.

Arguments	Descriptions
groupBase	This string is prepended to the basedn to construct the directory DN that contains group records.
groupMemberAttributes	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.
groupMemberFilter	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: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
groupNameAttribute	The group record attribute that contains the name of the group.
groupSearchFilter	This LDAP query is performed in the context of the groups directory to enumerate group records.
managerDn	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. **Note:** This user does not need to be an admin user with modify privileges.
managerPassword	If the managerDn property is set, this password is used to authenticate the manager user.
password	The password for the user that you are testing for this provider. The command will prompt for the password if it is omitted.
providerType	<ldap|activedirectory>
realm	This is an identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in "same name" collisions across multiple directory providers. The realm is appended to the user or group name when stored in the Commander server. For example, <*user*>@dir (where the realm is set to "dir").

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

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

Back to Top

API Commands - Email Configuration Management

createEmailConfig

Creates a new email configuration.

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

Arguments	Descriptions
`configName`	The name of your email configuration.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`mailFrom`	The email address used as the email sender address for notifications.
`mailHost`	The name of the email server host.
`mailPort`	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.
`mailProtocol`	This is either SSMTP or SMTP (not case-sensitive). The default is SMTP.
`mailUser`	This can be an individual or a generic name like "Commander" - name of the email user on whose behalf Commander sends email notifications.
`mailUserPassword`	Password for the email user who is sending notifications.

Positional arguments

`configName`

Response

None or 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: `ectool 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 conf
ig object"
```

Back to Top

deleteEmailConfig

Deletes an email configuration.

You must specify a `configName`.

Arguments	Descriptions
`configName`	The name of the email configuration you want to delete.

Positional arguments

`configName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteEmailConfig <configName>`

Example

```
ectool deleteEmailConfig emailA
```

Back to Top

getEmailConfig

Retrieves an email configuration by name.

You must specify a `configName`.

Arguments	Descriptions
`configName`	The name of the email configuration.

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

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

ectool

syntax: `ectool getEmailConfig <configName>`

Example

```
ectool getEmailConfig EmailConfig_test
```

Back to Top

getEmailConfigs

Retrieves all email configurations.

Arguments	Descriptions
None	

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
```

Back to Top

modifyEmailConfig

Modifies an existing email configuration.

You must specify the `configName`.

Arguments	Descriptions
`configName`	The name of your email configuration.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`mailFrom`	The email address used as the email "sender" address for notifications.
`mailHost`	The name of the email server host.

Arguments	Descriptions
mailPort	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.
mailProtocol	This is either SSMTP or SMTP (not case-sensitive). Default is SMTP.
mailUser	The name of the email user, which can be an individual or a generic name like "Commander".
mailUserPassword	The password for the email user.
newName	Supply any name of your choice to rename the email configuration.

Positional arguments

configName

Response

None or a status OK message.

ec-perl

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

Example

```
$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

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.

Arguments	Descriptions
`condition`	Only send mail if the condition evaluates to "true". The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is "false" or "0". If no condition is specified, the notification is ALWAYS sent.
`configName`	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 "ordered" Commander entities). **Note:** If using this argument, you must include either the `formattingTemplate` or the `formattingTemplateFile` argument also.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`destinations`	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.
`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	Descriptions
formattingTemplateFile	**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 "formatting template" field. This is an alternative argument for --formattingTemplate and is useful if the "formatting template" field spans multiple lines.
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.
projectName	The name of the project. **Also requires** procedureName
procedureName	The name of the procedure. **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
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.

Positional arguments

notifierName

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->createEmailNotifier("testNotifier",
        {eventType => "onStart",
         condition => "$[/javascript if(myJobStep.outcome == 'warning') 'true'; els
e '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.out
come
== 'warning') 'true'; else 'false';]"
    --destinations "user1@abc.com user2@abc.com emailAlias1@abc.com"
    --configName EmailConfig_test --formattingTemplate "Notification: Job:
$[/myJob/jobName]
    $[/myEvent/type] Job: $[/myJob/jobName] $[/myEvent/type] at $[/myEvent/time]"
    --projectName Project_test
    --procedureName Procedure_test
    --description "This is a test email notifier for Job completion"
```

Back to Top

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.

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.
`jobStepId`	The unique identifier for a job step, assigned automatically when the job step is created.
`notifierName`	The name of the email notifier you want to delete.
`procedureName`	The name of the procedure that contains this email notifier. **Also requires** `projectName`
`projectName`	The name of the project that contains this email notifier.
`stateDefinitionName`	The name of the state definition.
`stateName`	The name of the state.

Arguments	Descriptions
stepName	The name of the step that contains this email notifier. **Also requires** projectName and procedureName
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.

Positional arguments

notifierName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteEmailNotifier <notifierName> ...`

Example

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

Back to Top

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	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.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
notifierName	The name of your email notifier.

Arguments	Descriptions
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

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

ectool

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

Example

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

Back to Top

getEmailNotifiers

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

You must specify one or more object locators.

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.
jobStepId	The unique identifier for a job step, assigned automatically when the job step is created.
procedureName	The name of the procedure containing the email notifier. **Also requires** the projectName
projectName	The name of the project containing the 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

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
```

Back to Top

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`.

Arguments	Descriptions
`condition`	Only send mail if the condition evaluates to "true ". The condition is a string subject to property expansion. Notification will NOT be sent if the expanded string is "false" or "0". If no condition is specified, the notification is *always* sent.
`configName`	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 "ordered" ElectricCommander entities). **Note:** If using this argument, you **must** include either `formattingTemplate` or `formattingTemplateFile` also (not both arguments).
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`destinations`	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. **Note:** This argument is mandatory for the "create" operation.
`eventType`	<`onStart|onCompletion`> "onStart" triggers an event when the job/jobstep 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	Descriptions
formattingTemplateFile	**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 "formatting template" field. This is an alternative argument for formattingTemplate and is useful if the "formatting template" field spans multiple lines.
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.
newName	Supply any name of your choice to rename the email notifier.
notifierName	The name of your email notifier.
procedureName	The name of the procedure. **Also requires** projectName
projectName	The name of the project. **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

None or a status OK message.

ec-perl

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

Example

```
$cmdr->modifyEmailNotifier("testNotifier",
        {eventType => "onCompletion",
```

```
        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.

Arguments	Descriptions
`configName`	The name of the email configuration to use. If no configuration is specified, the configuration named "default" will be used. **Note:** The user must have "execute" permission on the configuration.
`subject`	The subject of the email message.
`to`	A "To" 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.
`cc`	A "Cc" 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.
`bcc`	A "Bcc" 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.
`header`	An RFC822 email header line (for example: "reply-to: user@host.com"). This option can be specified multiple times.

Arguments	Descriptions
`html`	The body of a simple HTML message.
`htmlFile`	Reads the specified client-side file and uses it as the body of a simple HTML message.
`text`	The body of a simple text message.
`textFile`	Reads the specified client-side file and uses it as the body of a simple text message.
`raw`	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.
`rawFile`	Reads the specified client-side file and uses it as the entire mail message, including headers.
`attachment`	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.
`inline`	`<contentId>=<fileName> [<contentId>=<fileName> ...]` 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 `cid:protocol`. For example: `<img src=cid:`*`myImage`*`">` could reference `--inline`*`myImage=image.jpg`* This option can be specified multiple times.
`mailFrom`	The "From" header to use when sending mail. Overrides the value from the email configuration if specified.
`mailHost`	The name of the mail server to use if no `configName` is specified. Overrides the value from the email configuration if specified.
`mailPort`	The mail server port to use if no `configName` is specified. Overrides the value from the email configuration if specified.
`mailProtocol`	The mail protocol. Must be either SMTP or SMTPS. Overrides the value from the email configuration if specified.
`mailUser`	The user account to use when authenticating to the mail server. Overrides the value from the email configuration if specified.
`mailUserPassword`	The password to use when authenticating to the mail server. Overrides the value from the email configuration if specified.

Arguments	Descriptions
multipartMode	<none|mixed|related|mixedRelated> Sets the multipart mode. Must be one of the following allowed multipart modes: `none` - non-multipart message `mixed` - single-root multipart element of type "mixed". Texts, inline elements, and attachments will be all be added to this root element. `related` - multipart message with a single root multipart element of type "related". Texts, inline elements, and attachments will be added to this root element. Works on most mail clients, except Lotus Notes. `mixedRelated` - multipart element "mixed" plus a nested multipart element of type "related". Texts and inline elements will be added to the nested "related' element, while attachments will be added to the "mixed' root element. Works on most mail clients other than Mac Mail and some situations on Outlook. If you experience problems, try "related". **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

```
$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

```
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


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

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.

ec-perl

Syntax:

```
$<object>->createEnvironment(<projectName>, <environmentName>,
 {<optionals>});
```

Example:

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

ectool

Syntax:

```
ectool createEnvironment <projectName> <environmentName>
[optionals...]
```

Example:

```
ectool 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.

ec-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 App1 ComponentA ResourceA
Artifact1 V3 --description aDescription
```

deleteEnvironment

Deletes 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

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

`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

Optional Arguments

None

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->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:

```
ectool 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");
```

ectool

Syntax:

```
ectool getEnvironments <projectName>
```

Example:

```
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:

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

Example:

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

ectool

Syntax:

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

Example:

```
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.

Argument Type: String

`environmentName`

Description: Name of the environment.

Argument Type: String

`applicationName`

Description: Name of the application.

Argument Type: String

Optional Arguments

None

Response

Retrieves a per-component grouped list of inventory items.

ec-perl

Syntax:

```
$<object>->getEnvironmentInventory(<projectName>, <environmentName>,
<applicationName>);
```

Example:

```
$ec->getEnvironmentInventory("Default", "aEnv", "App1");
```

ectool

Syntax:

```
ectool getEnvironmentInventory <projectName> <environmentName>
<applicationName>
```

Example:

```
ectool getEnvironmentInventory default newEnv App1
```

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

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

Retrieves an inventory item.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
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

`environmentEnabled`

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:

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

Example:

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

ectool

Syntax:

```
ectool modifyEnvironment <projectName> <environmentName>
[optionals...]
```

Example:

```
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.

ec-perl

Syntax:

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

Example:

```
$ec->modifyEnvironmentInventoryItem("Default", "aEnv", "App1", "Component1",
"Server1", "Artifact1", "V3");
```

ectool

Syntax:

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

Example:

```
ectool modifyEnvironmentInventoryItem default testEnv App1 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:

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

Example:

```
$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:

```
ectool deleteEnvironmentTier default newEnv  tierToDelete
```

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:

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

Example:

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

ectool

Syntax:

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

Example:

```
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:

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

Example:

```
$ec->getEnvironmentTiers("Default", "newEnv");
```

ectool

Syntax:

```
ectool getEnvironmentTiers <projectName> <environmentName>
```

Example:

```
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.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
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`.

Arguments	Descriptions
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`gatewayDisabled`	<*Boolean flag* - `0|1|true|false` > If set to 1, the gateway is disabled.
`gatewayName`	The name of the gateway. Supply any of your choice, but the name must be unique among other gateway names.
`hostName1`	The agent host name where *Resource1* resides. This host name is used by *Resource2* to communicate with *Resource1*. Do not specify this option is you want to use the host name from *Resource1's* definition.
`hostName2`	The agent host name where *Resource2* resides. This host name is used by *Resource1* to communicate with *Resource2*. Do not specify this option is you want to use the host name from *Resource2's* definition.
`port1`	The port number used by *Resource1* - defaults to the port number used by the resource.
`port2`	The port number used by *Resource2* - defaults to the port number used by the resource.

Arguments	Descriptions
resourceName1	The name of your choice for the first of two required gateway resources. Do **not** include "spaces" in a resource name.
resourceName2	The name of your choice for the second of two required gateway resources. Do **not** include "spaces" in a resource name.

Positional arguments

gatewayName

Response

Returns a gateway object.

ec-perl

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

Example

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

ectool

syntax: `ectool createGateway <gatewayName> ...`

Example

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

Back to Top

deleteGateway

Deletes a gateway.

You must supply a `gatewayName`.

Arguments	Descriptions
gatewayName	The name of the gateway to delete.

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"
```

Back to Top

getGateway

Finds a gateway by name.

You must specify a `gatewayName`.

Arguments	Descriptions
`gatewayName`	The name of the gateway you want to find.

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
```

Back to Top

getGateways

Retrieves all gateways.

Arguments	Descriptions
None	

Positional arguments

None.

Response

Returns one or more gateway elements.

ec-perl

syntax: `$cmdr->getGateways();`

Example

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

ectool

syntax: `ectool getGateways`

Example

```
ectool getGateways
```

Back to Top

modifyGateway

Modifies an existing gateway.

You must specify a `gatewayName`.

Arguments	Descriptions
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`gatewayDisabled`	<*Boolean flag* - `0|1|true|false` > If set to 1, the gateway is disabled.
`gatewayName`	The name of the gateway you want to modify.

Arguments	Descriptions
hostName1	The agent host name where *Resource1* resides. This host name is used by *Resource2* to communicate with *Resource1*. Do not specify this option is you want to use the host name from *Resource1's* definition.
hostName2	The agent host name where *Resource2* resides. This host name is used by *Resource1* to communicate with *Resource2*. Do not specify this option is you want to use the host name from *Resource2's* definition.
newName	Supply any name of your choice to rename the gateway.
port1	The port number used by *Resource1* - defaults to the port number used by the resource.
port2	The port number used by *Resource2* - defaults to the port number used by the resource.
resourceName1	The name of your choice for the first of two required gateway resources. Do **not** include "spaces" in a resource name.
resourceName2	The name of your choice for the second of two required gateway resources. Do **not** include "spaces" in a resource name.

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"
```

Back to Top

createZone

Creates a new zone.

You must specify a `zoneName`.

Arguments	Descriptions
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`zoneName`	The name of the zone. Supply any unique name of your choice.

Positional arguments

`zoneName`

Response

Returns a zone object.

ec-perl

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

Example

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

ectool

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

Example

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

Back to Top

deleteZone

Deletes an existing zone.

You must specify a `zoneName`.

Arguments	Descriptions
`zoneName`	The name of the zone to delete.

Positional arguments

zoneName

Response

None

ec-perl

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

Example

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

ectool

syntax: `ectool deleteZone <zoneName>`

Example

```
ectool deleteZone DevZone
```

Back to Top

getZone

Finds a zone by name.

You must specify a `zoneName`.

Arguments	Descriptions
zoneName	The name of the zone you want to find.

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>`

Example

```
ectool getZone DevZone
```

Back to Top

getZones

Retrieves all zones.

Arguments	Descriptions
None	

Positional arguments

None

Response

Returns a zone object.

ec-perl

syntax: `$cmdr->getZones();`

Example

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

ectool

syntax: `ectool getZones`

Example

```
ectool getZones
```

Back to Top

modifyZone

Modifies an existing zone.

You must specify a `zoneName`.

Arguments	Descriptions
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
newName	Supply any unique name of your choice to rename the zone.
zoneName	The name of this zone.

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 d
ev group uses."});
```

ectool

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

Example

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

Back to Top

API Commands - Job Management

External Job APIs

abortAllJobs

Aborts all running jobs.

Arguments	Descriptions
`force`	<*Boolean flag* - `0|1|true|false` > If set to 1, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, executing steps marked “always run".
`reason`	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 "for your reference only."

Positional arguments

None

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool abortAllJobs ...`

Example

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

Back to Top

abortJob

Aborts a running job.

You must supply a `jobId`.

Arguments	Descriptions
`force`	<*Boolean flag* - `0|1|true|false`> If set to 1, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, executing steps marked "always run".
`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.
`reason`	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 "for your reference only."

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
```

Back to Top

abortJobStep

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`.

Arguments	Descriptions
`force`	<*Boolean flag* - `0|1|true|false`> If set to 1, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, for example, executing steps marked "always run".
`jobStepId`	The `jobStep` to abort - the unique identifier for a job step, assigned automatically when the job step is created.
`reason`	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 "for your reference only."

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
```

Back to Top

deleteJob

Deletes a job from the ElectricCommander database.

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

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.

Arguments	Descriptions
jobId	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.
jobStepId	The unique identifier for a job step whose job steps you want to retrieve - assigned automatically when the job step is created.
filter	A list of zero or more filter criteria definitions used to define objects to find. See the findObjects API for complete description for using filters.

Arguments	Descriptions
numObjects	<full object count> 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.
select	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.

Positional arguments

jobId, or jobStepId

Response

One or more jobStep elements.

ec-perl

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

Example 1

```
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

```
my $xPath = $cmdr->findJobSteps({jobStepId  => "5da765dd-73f1-11e3-b67e-b0a42052415
3"});
   print "Return data from Commander:\n" .
        $xPath-> findnodes_as_string("/"). "\n";
```

ectool

Not supported.

Back to Top

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.

ec-perl

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
```

Back to Top

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.

ec-perl

syntax: `$cmdr->getJobInfo(<jobId>);`

Example

```
$cmdr->getJobInfo(4fa765dd-73f1-11e3-b67e-b0a420524153);
```

ectool

syntax: `ectool getJobInfo <jobId>`

Example

```
ectool getJobInfo 4fa765dd-73f1-11e3-b67e-b0a420524153
```

Back to Top

getJobNotes

Retrieves the notes property sheet from a job.

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

A `propertySheet` element that contains the job.

ec-perl

syntax: `$cmdr->getJobNotes(<jobId>);`

Example

```
$cmdr->getJobNotes(4fa765dd-73f1-11e3-b67e-b0a420524153);
```

ectool

syntax: `ectool getJobNotes <jobId>`

Example

```
ectool getJobNotes 4fa765dd-73f1-11e3-b67e-b0a420524153
```

Back to Top

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.

Arguments	Descriptions
`firstResult`	*<index number>* 0-based index identifies the first element returned from filtered, sorted result set.
`maxResults`	*<result count>* This number sets the maximum number of returned jobs.
`sortKey`	`<jobId|jobName|start|finish|procedureName>` Choose how you want to sort the list.
`sortOrder`	`<ascending|descending>`
`status`	`<running|completed|runnable>` Choose status to restrict the list.

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)?

```
ectool getJobs --maxResults 10
```

How do I get the next 10 jobs (index 10-19)?

```
ectool getJobs --firstResult 10 --maxResults 10
```

How do I get the most recent job by start time?

```
ectool getJobs --sortKey start --sortOrder descending --maxResults 1
```

Back to Top

getJobsForSchedule

Retrieves jobs started by a specific schedule.

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

Arguments	Descriptions
`projectName`	The name of the project that contains this schedule.
`scheduleName`	The name of the schedule that launched these jobs.

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: `$cmdr->getJobsForSchedule(<projectName>, <scheduleName>);`

Example

```
$cmdr->getJobsForSchedule('Test', 'ea1');
```

ectool

syntax: `ectool getJobsForSchedule <projectName> <scheduleName>`

Example

```
ectool getJobsForSchedule Test ea1
```

Back to Top

getJobStatus

Retrieves the status of a job.

You must specify the `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

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
```

Back to Top

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`.

Arguments	Descriptions
`jobStepId`	The unique identifier for a job step, assigned automatically when the job step is created.
`structureOnly`	<*Boolean flag* - <`0|1|true|false`> Reduces the amount of information returned to minimal structural information.

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
```

Back to Top

getJobStepStatus

Retrieves the status of a job step.

You must specify the `jobStepId`.

Arguments	Descriptions
`jobStepId`	The unique identifier for a job step, assigned automatically when the job step is created.

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
```

Back to Top

moveJobs

Moves jobs from one project to another project.

You must specify `sourceProject` and `destinationProject`.

Arguments	Descriptions
`sourceProject`	The name of the project that contains the jobs you want to move.
`destinationProject`	The new project that will contain the jobs.

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"
```

Back to Top

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`.

Arguments	Descriptions
`actualParameter`	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. Used in conjunction with `procedureName` to set the value of the actual parameters. Do not use this argument with `scheduleName`.
`credential`	Use the following syntax to specify a credential: `<credName>=<userName> [<credName>=<userName> ...]]`
`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	Descriptions
Note: The following two arguments are used to control whether runProcedure returns immediately or waits until the job completes. *** If `pollInterval` is used and `timeout` is not used, `pollInterval` will timeout in 60 seconds.	
`pollInterval`	If this option is not specified, `runProcedure` returns immediately. If it is specified, `runProcedure` waits until the job completes. This argument requires setting a value in *seconds* to determine how often ectool queries the Commander server for job status, but this is not an indefinite activity - set the `timeout` value to extend the `pollInterval` for longer than 60 seconds if needed.
`timeout`	This argument requires a value set in *seconds*. If `pollInterval` is specified, this timeout causes `runProcedure` to stop waiting for the job to complete. It does not stop the job itself. If `pollInterval` is used and `timeout` is not used, `pollInterval` will timeout in 60 seconds.

Positional arguments

`projectName, procedureName, scheduleName`

Response

The new job ID number.

ec-perl

syntax: `$cmdr->runProcedure(<project name>, {<optionals});`

Example

```
$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

```
ectool runProcedure <project name> --procedureName <procedure name>
   --scheduleName <schedule name>

ectool runProcedure "Sample Project" --procedureName "Delay"
```

```
--actualParameter "Delay Time=10"
```

Back to Top

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.

Arguments	Descriptions
`jobId`	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.
`newName`	Supply any name of your choice to rename the job.

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 th
e 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 c
ommand line)

ectool setJobName "TestJob"_252
```

Back to Top

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`.

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.
`force`	<*Boolean flag* - `0|1|true|false`> If true, all external steps belonging to the job will be marked "complete". Determines whether all external steps under the job should be recursively marked "complete". **Note:** If this API is called on a job launched with `runProcedure`, there is no effect unless `force` is enabled, in which case only external steps are affected.

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

```
$cmdr->completeJob(1234);
```

ectool

syntax: `ectool completeJob <jobId>`

Example

```
ectool completeJob 1234
```

Back to Top

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`.

Arguments	Descriptions
exitCode	The step's exit code.
force	<*Boolean flag* - `0|1|true|false`> If true, all external steps under the job should be recursively marked "complete". **Note:** If this API is called on a job launched with `runProcedure`, there is no effect unless `force` is enabled, in which case only external steps are affected.

Arguments	Descriptions
jobStepId	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.
outcome	Possible values for outcome: success - The job step finished successfully. warning - The job step completed with no errors, but encountered some suspicious conditions. error - The job step has finished execution with errors. skipped - The job step was skipped.

Positional arguments

jobStepId

Response

None or status OK message.

ec-perl

syntax: `$cmdr->completeJobStep(<jobStepId>);`

Example

```
$cmdr->completeJobStep(5da765dd-73f1-11e3-b67e-b0a420524153);
```

ectool

syntax: `ectool completeJobStep <jobStepId>`

Example

```
ectool completeJobStep 5da765dd-73f1-11e3-b67e-b0a420524153
```

Back to Top

createJob

Creates an externally managed job that will serve as a container for external job steps.

You must specify projectName or destinationProject.

Arguments	Descriptions
destinationProject	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.

Arguments	Descriptions
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 ...`

Example

```
ectool createJob --projectName "Sample Project"
```

Back to Top

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`.

Arguments	Descriptions
`actualParameter`	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.
`alwaysRun`	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 "cleanup" step that should run whether the job step is successful or not. The value for `alwaysRun` is a <*Boolean flag* - `0|1|true|false`>. Defaults to "false".
`broadcast`	Use this flag to run the same job step on several resources at the same time. The job step is "broadcast" to all resources listed in the `resourceName` argument. The `broadcast` value = <*Boolean flag* -`0|1|true|false`>. This argument is applicable only to command job steps. Defaults to "false".
`command`	The command to run. This argument is applicable to command job steps only.
`condition`	If empty or non-zero, the job step will run. If set to "0", 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.
`credential`	Refers to one or more credentials to attach to this job step. These are "dynamic" 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.

Arguments	Descriptions
credentialName	The credential to use for impersonation on the agent. 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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
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 job 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".

Arguments	Descriptions
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 step. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" 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. • Step - keeps the resource for the duration of the job step. • Call - keeps the resource for the duration of the procedure that called this job step, which is equivalent to 'job' for top level steps.
external	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. 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 <*Boolean flag* - 0|1|true|false>. Default is "false".
jobStepId	ID of the parent job step. If both jobStepId and parentPath are specified, parentPath is preferred.
jobStepName	The name of the job step. You can use any name of your choice.
logFileName	A custom log file name produced by running the job step. By default, Commander assigns a unique name for this file.
parallel	If set, indicates this job step should run at the same time as adjacent job steps marked to run as parallel also. The value for parallel is a <*Boolean flag* -0|1|true|false>. Defaults to "false".
parentPath	The path of the parent job step. If both jobStepId and parentPath are specified, parentPath is preferred.
postProcessor	The name of a program to run after a job step completes. This program looks at the job step output to find errors and warnings. Commander includes a customizable program called "postp" 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).

Arguments	Descriptions
precondition	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, `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]`
procedureName	The name of the procedure that will contain this job step.
projectName	The name of the project that contains the procedure where you are adding a new job step.

Arguments	Descriptions
releaseExclusive	<*Boolean flag* - 0|1|true|false> Declares whether or not this job step will release its resource, which is currently held "exclusively". **Note:** Setting this flag to "true" is the same as setting releaseMode to release.
releaseMode	Use one of the following options: • None - the "default" - no action if the resource was not previously marked as "retain". • Release - releases the resource at the end of this job step. If the resource for the job step was previously acquired with "Retain exclusive" (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. • Release to job - allows a job step to promote a "step exclusive" resource to a Job exclusive resource.
resourceName	The name of the resource you want this job step to use.
shell	Where *shell* is the name of a program used to execute commands contained in the "command" 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.
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 modifyJobStep to update the status. Defaults to pending. 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.
stepName	The name of the new job step you are creating. You can use any name of your choice.
subprocedure	The name of the nested procedure to call when this job step runs. If a subprocedure is specified, do not include the command or commandFile options.

Arguments	Descriptions
subproject	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.
timeLimit	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.
timeLimitUnits	Specify hours|minutes|seconds for time limit units.
workingDirectory	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. **Note:** If running a job step on a proxy resource, this directory must exist on the proxy target.
workspaceName	The name of the workspace where this job step's log files will be stored.

Positional arguments

jobStepId or parentPath

Response

Returns a jobStep object.

ec-perl

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

Examples

```
$cmdr->createJobStep ({parentPath => "/jobs/123", external => "1"});

$cmdr->createJobStep ({jobStepId => "5da765dd-73f1-11e3-b67e-b0a420524153", externa
l => "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 bat
ch 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'
                     }
                ],
            }
        ),
    );

    # 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
```

Back to Top

modifyJob

Modifies the status of an externally managed job.

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.

Arguments	Descriptions
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 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

jobId

Response

The jobId element.

ec-perl

syntax: `$cmdr->modifyJob (<jobId>, {<optionals>});`

Example

```
$cmdr->modifyJob (4fa765dd-73f1-11e3-b67e-b0a420524153, {status => "running"});
```

ectool

syntax: `ectool modifyJob <jobId> ...`

Example

```
ectool modifyJob 4fa765dd-73f1-11e3-b67e-b0a420524153 --status "running"
```

Back to Top

modifyJobStep

Modifies the status of an externally managed job step.

You must specify a projectName and jobStepId.

Arguments	Descriptions
jobStepId	The Commander-generated ID for the job step.

Arguments	Descriptions
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: `$cmdr->modifyJobStep (<jobStepId>, {<optional>});`

Example

```
$cmdr->modifyJobStep (4fa765dd-73f1-11e3-b67e-b0a420524153, {status => "running"});
```

ectool

syntax: `ectool modifyJobStep <jobStepId> ...`

Example

```
ectool modifyJobStep 4fa765dd-73f1-11e3-b67e-b0a420524153 --status "running"
```

Back to Top

waitForJob

Waits until the specified job reaches a given status or the timeout expires. Returns the result from the final getJobStatus query.

Arguments	Descriptions
jobId	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.
finalStatus	The status to wait for. Must be either "running" or "completed" (default is "completed").
timeout	The number of seconds to wait before giving up on a request.

Positional arguments

`jobId`

Response

Returns the result from the final `getJobStatus` query.

ec-perl

syntax:`$cmdr->waitForJob ($4fa765dd-73f1-11e3-b67e-b0a420524153, {<optional>);`

Example

```
$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`.

Arguments	Descriptions
`formalParameterName`	The name of the procedure's parameter, containing a credential reference.
`procedureName`	The name of the procedure containing the step.
`projectName`	The name of the project containing the step.
`stepName`	The name of the step to attach the parameter credential.

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 Credentia
l"});
```

ectool

syntax: `ectool attachParameter <projectName> <procedureName> <stepName> <formalParameterName>`

Example

```
ectool attachParameter "Test Proj" "Run Build" "Get Sources" "SCM Credential"
```

Back to Top

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:

```
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:

```
# Run the procedure
      $xPath = $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:

```
{ ..., => ..., 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:

```
actualParameter => {actualParameterName => 'parm1',
                                  value => 'myval'}
```

You must specify `projectName, procedureName, stepName, and actualParameterName`.

Arguments	Descriptions
`actualParameterName`	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.
`procedureName`	The name of the procedure containing the step.
`projectName`	The name of the project containing the procedure.
`scheduleName`	The name of the schedule containing this parameter.
`stateDefinitionName`	The name of the state definition.
`stepName`	The name of the step that calls a subprocedure.
`transitionDefinitionName`	The name of the transition definition.
`value`	This value is passed to the subprocedure as the value of the matching formal parameter.
`workflowDefinitionName`	The name of the workflow definition.

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"
```

Back to Top

createFormalParameter

Creates a new formal parameter.

Arguments	Descriptions
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`defaultValue`	This value is used for the formal parameter if a value is not supplied by the caller.
`expansionDeferred`	<*Boolean flag* - `0|1|true|false`> Default is "false," which means the formal parameter is expanded immediately.
`formalParameterName`	The name for this parameter - used when the procedure is invoked to specify a value for the parameter.

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/select", "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");
```

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>`

Back to Top

deleteActualParameter

Deletes an actual parameter.

You must specify a `projectName`, `procedureName`, `stepName`, and `actualParameterName`.

Arguments	Descriptions
`actualParameterName`	The name of the actual parameter you want to delete.
`procedureName`	The name of the procedure that contains the step with this parameter.
`projectName`	The name of the project that contains this actual parameter.
`scheduleName`	The name of the schedule containing the actual parameter.

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 Par
m');
```

ectool

syntax: `ectool deleteActualParameter <projectName> <procedureName> <stepName> <actualParameterName>`

Example

```
ectool deleteActualParameter "Sample Project" "CallSub" "Step1" "Different Parm"
```

Back to Top

deleteFormalParameter

Deletes a formal parameter.

You must specify `projectName`and `formalParameterName`.

Arguments	Descriptions
formalParameterName	The name of the formal parameter you want to delete.

Arguments	Descriptions
procedureName	The name of the procedure that contains this 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 that contains the procedure/parameter you want to delete.
stateDefinitionName	The name of the state definition.
workflowDefinitionName	The name of the workflow definition.

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"
```

Back to Top

detachParameter

Detaches a formal parameter from a step.

You must specify `projectName`, `procedureName`, `stepName`, and `formalParameterName`.

Arguments	Descriptions
`formalParameterName`	The name of the parameter to detach.
`procedureName`	The name of the procedure that contains this parameter.
`projectName`	The name of the project that contains this parameter.
`stepName`	The name of the step where this parameter is currently attached.

Positional arguments

```
projectName, procedureName, stepName, formalParameterName
```

Response

None or a status OK message.

ec-perl

syntax:
```
$cmdr->detachParameter(<projectName>, <procedureName>, <stepName>,
          <formalParameterName>);
```

Example

```
$cmdr-> detachParameter("Test Proj", "Run Build", "Get Sources", "SCM Credential");
```

ectool

syntax:
```
ectool detachParameter <projectName> <procedureName> <stepName>
          <formalParameterName>
```

Example

```
ectool detachParameter "Test Proj" "Run Build" "Get Sources" "SCM Credential"
```

Back to Top

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	Descriptions
actualParameterName	The name of the actual parameter.
applicationName	The name of the application, if the actual parameter is on an application process step; must be unique among all projects.
componentName	The name of the component, if the actual parameter is on a component process step.
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. Supply this argument to query a subprocedure call to the job step's parameter.
procedureName	The name of the procedure to query for the procedure step's parameter.
processName	The name of the process, if the actual parameter is on a process step.
processStepName	The name of the process step, if the actual parameter is on a process step.
projectName	The name of the project to query for a schedule or procedure step's parameter.
scheduleName	The name of the schedule to query for the schedule's parameter.
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step to query for the step's parameter.
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.

Positional arguments

actualParameterName

Response

One actualParameter element.

ec-perl

syntax: `$cmdr->getActualParameter(<actualParameterName>, {...});`

Example

```
$cmdr->getActualParameter("Extra Parm",
    {"projectName" => "Sample Project",
   "procedureName" => "CallSub",
        "stepName" => "Step1"});
```

ectool

syntax: `ectool getActualParameter <actualParameterName> ...`

Example

```
getActualParameter "Extra Parm" --projectName "Sample Project"
  --procedureName "CallSub" --stepName "Step1"
```

Back to Top

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.

Arguments	Descriptions
`applicationName`	The name of the application, if the actual parameter is on an application process step; must be unique among all projects.
`componentName`	The name of the component, if the actual parameter is on a component process step.
`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.
`projectName`	The name of the project containing these parameters.
`procedureName`	The name of the procedure containing these parameters.
`processName`	The name of the process, if the actual parameter is on a process step.
`processStepName`	The name of the process step, if the actual parameter is on a process step.

Arguments	Descriptions
scheduleName	The name of the schedule containing parameters.
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing parameters.
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.

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

```
$cmdr-> getActualParameters({"projectName" => "Sample Project",
  "procedureName" => "CallSub",
       "stepName" => "Step1"});
```

ectool

syntax: `ectool getActualParameters ...`

Example

```
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`.

Arguments	Descriptions
formalParameterName	The name of the formal parameter.
procedureName	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.
projectName	The name of the project containing the procedure.
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.

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.

Arguments	Descriptions
procedureName	The name of the procedure. **Also requires** the projectName
projectName	The name of the project containing the object whose parameters are being retrieved.
scheduleName	The name of the schedule. **Also requires** the projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step. **Also requires** the projectName and procedureName
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.

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.

Arguments	Descriptions
actualParameterName	The name of the actual parameter to modify.
newName	Supply a name of your choice to rename the parameter.
procedureName	The name of the procedure containing the step with this parameter.
projectName	The name of the project containing this parameter.
scheduleName	The name of the schedule.
stateDefinitionName	The name of the state definition.
stepName	The name of the step containing this parameter.
transitionDefinitionName	The name of the transition definition.
value	Changes the current value on an actual parameter. This value is passed to the subprocedure as the value of the matching formal parameter.
workflowDefinitionName	The name of the workflow definition.

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->modifyActualParameter(<projectName>, <procedureName>, <stepName>, <actualParameterName>, {<optionals>});`

Example

```
$cmdr->modifyActualParameter("Sample Project", "CallSub", "Step1", "Extra Parm",
    {newName => "myParm"});
```

ectool

syntax: `ectool modifyActualParameter <projectName> <procedureName> <stepName> <actualParameterName> ...`

Example

```
ectool modifyActualParameter "Sample Project" "CallSub" "Step1" "Extra Parm"
   --newName "Different Parm"
```

Back to Top

modifyFormalParameter

Modifies an existing formal parameter.

Arguments	Descriptions
defaultValue	This value is used for he formal parameter if one is not supplied by the caller.
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
expansionDeferred	<*Boolean flag* - 0|1|true|false> Default is "false," which means the formal parameter is expanded immediately.
formalParameterName	The name for this formal parameter. Used when the procedure is invoked to specify a value for the parameter.
newName	Supply any name of your choice to rename the parameter.
procedureName	The name of the procedure containing this 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 this parameter.
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	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 a status OK message.

ec-perl

syntax: `$cmdr->modifyFormalParameter(<projectName>, <formalParameterName>, {<optionals>});`

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

syntax:`$cmdr->modifyFormalParameter(<projectName>, <procedureName>, <formalParameterName>, {<optionals>});`

Example

```
$cmdr->modifyFormalParameter("Sample Project", "Branch Name",
   {defaultValue => "main"});
```

ectool

For procedure parameters:

syntax: `ectool modifyFormalParameter <projectName> <formalParameterName> ...`

For backward compatibility with releases earlier than ElectricCommander 5.0, you can also enter:

syntax: `ectool modifyFormalParameter <projectName> <procedureName> <formalParameterName> ...`

Example

```
ectool modifyFormalParameter "Sample Project" "Branch Name"
   --defaultValue main
```

For workflow state parameters:

syntax: `ectool modifyFormalParameter --formalParameterName <name> --projectName <name> --workflowDefinitionName <name> --stateDefinitionName <name>`

Back to Top

API Commands - Plugin Management

deletePlugin

Deletes an existing plugin object without deleting the associated project or files.

You must specify a `pluginName`.

Arguments	Descriptions
`pluginName`	The name of the plugin you want to delete.

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
```

Back to Top

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
```

Back to Top

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();
```

ectool

syntax: `ectool getPlugins`

Example

```
ectool getPlugins
```

Back to Top

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`.

Arguments	Descriptions
`force`	<*Boolean flag* - `0|1|true|false`> Specifying false causes an existing plugin with the same key and version to be overwritten with the new plugin contents, otherwise an error is returned.
`url`	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 `http://url`), the server will download it. If the location is a `file:` reference, the file will be read directly from the specified location on the server's file system.

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`.

Arguments	Descriptions
`author`	The author of the plugin.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`label`	The name of the plugin as displayed on the Plugins web page.
`pluginName`	The name of the plugin to modify. If the name is specified without a version number, the currently promoted version is used if possible.

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`.

Arguments	Descriptions
`pluginName`	The name of the plugin to promote. If the name is specified without a version number, the currently promoted version is used if possible.
`promoted`	<*Boolean flag* - `0|1|true|false`> The new value of the promoted flag for the specified plugin. Default is "true", which means the plugin will be promoted. If you want to demote the plugin, use the value of `"0"` or false.

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: `$cmdr->promotePlugin(<pluginName>, {<optionals});`

Example

```
$cmdr->promotePlugin("TheWidget-1.0");
```

ectool

syntax: `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`.

Arguments	Descriptions
pluginName	The name of the plugin to uninstall. If the name is specified without a version number, the currently promoted version is used if possible.
timeout	The maximum amount of time to spend waiting for this operation to complete.

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

Creates a new procedure for an existing project.

You must specify `projectName` and `procedureName`.

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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`jobNameTemplate`	Template used to determine the default name of jobs launched from a procedure.
`procedureName`	The name you define for this procedure. You can use any name of your choice.
`projectName`	The name of the project that contains this procedure.
`resourceName`	The name of a resource or pool to use as the default for steps run by this procedure.

Arguments	Descriptions
`timeLimit`	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 "timer" for the procedure starts as soon as the calling step/job becomes runnable (all preconditions are satisfied).
`timeLimitUnits`	Time limit units are `hours|minutes|seconds`
`workspaceName`	The name of the workspace to use as the default for steps run by this procedure.

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 Resourc
e"});
```

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`.

Arguments	Descriptions
`actualParameter`	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.
`alwaysRun`	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 "cleanup" step that should run whether the job is successful or not. The value for `alwaysRun` is a <*Boolean flag* -`0|1|true|false`>. Defaults to "false".
`broadcast`	Use this flag to run the same step on several resources at the same time. The step is "broadcast" to all resources listed in the `resourceName` argument. The `broadcast` value = <*Boolean flag* -`0|1|true|false`>. This argument is applicable only to command steps. Defaults to "false".
`command`	The command to run. This argument is applicable to command steps only.
`commandFile`	**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 "command" field. This is an alternative argument for `command` and is useful if the "command" field spans multiple lines. The `commandFile` value is the actual *command file* text. This argument is applicable to command steps only.
`condition`	If empty or non-zero, the step will run. If set to "0", 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.
`credentialName`	The credential to use for impersonation on the agent. `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.

Arguments	Descriptions
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
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.
logFileName	A custom log file name produced by running the step. By default, ElectricCommander assigns a unique name for this file.
parallel	If set, indicates this step should run at the same time as adjacent steps marked to run as parallel also. The value for parallel is a <*Boolean flag* -0|1|true|false>. Defaults to "false".

Arguments	Descriptions
postProcessor	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 "postp" 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).
precondition	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 \"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]
procedureName	The name of the procedure that will contain this step.
projectName	The name of the project that contains the procedure where you are adding a new step.
releaseExclusive	<*Boolean flag* - 0|1|true|false> Declares whether or not this step will release its resource, which is currently held exclusively. **Note:** Setting this flag to "true" is the same as setting releaseMode to release.

Arguments	Descriptions
releaseMode	Use one of the following options: • None - the "default" - no action if the resource was not previously marked as "retain". • Release - releases the resource at the end of this step. If the resource for the step was previously acquired with "Retain exclusive" (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. • Release to job - allows a step to promote a "step exclusive" resource to a Job exclusive resource.
resourceName	The name of the resource you want this step to use.
shell	Where *shell* is the name of a program used to execute commands contained in the "command" 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.
stepName	The name of the new step you are creating. You can use any name of your choice.
subprocedure	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.
subproject	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.
timeLimit	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	Specify hours|minutes|seconds for time limit units.
workingDirectory	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. **Note:** If running a step on a proxy resource, this directory must exist on the proxy target.

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:

```
<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:

```
{... => ..., 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:

```
actualParameter => {actualParameterName => 'parm1', value => 'myval'}
```

Example

```
$cmdr->createStep("Test Proj", "Run Build", "Common Cleanup", {subprocedure => "Del
ay",
```

```
    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"
```

Back to Top

deleteProcedure

Deletes a procedure, including all steps.

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

Arguments	Descriptions
`procedureName`	The name of the procedure you want to delete.
`projectName`	The name of the project that contains this procedure.

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"
```

Back to Top

deleteStep

Deletes a step from a procedure.

You must specify `projectName`, `procedureName`, and `stepName`.

Arguments	Descriptions
`procedureName`	The name of the procedure that contains this step.
`projectName`	The name of the project that contains this procedure/step.
`stepName`	The name of the step you want to delete.

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"
```

Back to Top

getProcedure

Finds a procedure by its name.

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

Arguments	Descriptions
`procedureName`	The name of the procedure you are retrieving.
`projectName`	The name of the project containing the procedure to retrieve.

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"
```

Back to Top

getProcedures

Retrieves all procedures in one project.

You must specify the `projectName`.

Arguments	Descriptions
projectName	The name of the project containing the procedures to retrieve.

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`.

Arguments	Descriptions
`procedureName`	The name of the procedure that contains the step.
`projectName`	The name of the project where you want to find a step.
`stepName`	The name of the step.

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`.

Arguments	Descriptions
`procedureName`	The name of the procedure that contains the steps.
`projectName`	The name of the project containing the procedure for the steps you want to find.

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"
```

Back to Top

modifyProcedure

Modifies an existing procedure.

You must specify projectName and procedureName.

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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
jobNameTemplate	Job name format for jobs created by running this procedure.
newName	Supply any name of your choice to rename the procedure.
procedureName	The name of the procedure to modify.

Arguments	Descriptions
projectName	The name of the project to modify. **Also requires** procedureName
resourceName	The name of the default resource where steps belonging to this procedure will run. This name may be a resource pool name.
timeLimit	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 "timer" for the procedure starts as soon as the calling step/job becomes runnable (all preconditions are satisfied).
timeLimitUnits	Time limit units are hours|minutes|seconds
workspaceName	The name of the default workspace where job output is stored.

Positional arguments

projectName, procedureName

Response

None or a status OK message.

ec-perl

syntax: $cmdr->modifyProcedure(<projectName>, <procedureName>, {...});

Example

```
$cmdr->modifyProcedure("Test Proj", "Run Build", {resourceName =>
   "Windows - Bldg. 11"});
```

ectool

syntax: ectool modifyProcedure <projectName> <procedureName> ...

Example

```
ectool modifyProcedure "Test Proj" "Run Build"
   --resourceName "Windows - Bldg. 11"
```

Back to Top

modifyStep

Modifies an existing step.

You must specify projectName, procedureName, and stepName.

Arguments	Descriptions
actualParameter	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.
alwaysRun	<*Boolean flag* - 0|1|true|false> 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 "cleanup" step that should run whether the job is successful or not.
broadcast	<*Boolean flag* - 0|1|true|false> Use this flag to run the same step on several resources at the same time. The step is "broadcast" to all resources listed in the resourceName.
clearActualParameters	<*Boolean flag* - 0|1|true|false> If set to true, all actual parameters will be removed from the step.
command	The step command.
commandFile	**This option is supported only in Perl and ectool 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.
condition	If empty or non-zero, the step will run. If set to "0", 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.
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>

Arguments	Descriptions
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.
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	<*Boolean flag* - 0|1|true|false> Indicates if this step should run at the same time as adjacent steps marked to run as parallel also.

Arguments	Descriptions
precondition	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 \"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]`
procedureName	The name of the procedure containing the step to modify. **Also requires** `projectName`
projectName	The name of the project containing the step to modify. **Also requires** `procedureName`
postProcessor	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 "postp" for this purpose.
releaseExclusive	<*Boolean flag* - `0|1|true|false`> Declares whether or not this step will release its resource, which is currently held exclusively. **Note:** Setting this flag to "true" is the same as setting `releaseMode` to `"release"`.

Arguments	Descriptions
releaseMode	Use one of the following options: • None - the "default" - no action if the resource was not previously marked as "retain". • Release - releases the resource at the end of this step. If the resource for the step was previously acquired with "Retain exclusive" (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. • Release to job - allows a step to promote a Step exclusive resource to a Job exclusive resource.
resourceName	The name of the resource used by this step.
shell	Where *shell* is the name of a program used to execute commands contained in the "command" 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.
stepName	The name of the step. **Also requires** projectName and procedureName
subprocedure	The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the command or commandField.
subproject	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.
timeLimit	The maximum length of time the step is allowed to run. After the time specified, the step will be aborted.
timeLimitUnits	<hours|minutes|seconds>
workingDirectory	The Commander agent sets this directory as the "current working directory," 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.
workspaceName	The name of the workspace used by this step.

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`.

Arguments	Descriptions
`beforeStep`	Moves the step (stepName) to position before the step "named" by this option. If omitted, `stepName` is moved to the end of the list of steps.
`procedureName`	The name of the procedure containing the step to move.
`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 => "Compil
e"});
```

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

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.

ec-perl

Syntax:

```
$<object>->createProcess(<projectName>, <processName>, {<optionals>});
```

Example:

```
$ec->createProcess("default", "process1", {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool createProcess <projectName> <processName> [optionals...]
```

Example:

```
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.

ec-perl

Syntax:

```
$<object>->deleteProcess(<projectName>, <processName>, {<optionals>});
```

Example:

```
$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

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 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
```

getProcesses

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: 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"});
```

ectool

Syntax:

```
ectool modifyProcess <projectName> <processName> [optionals...]
```

Example:

```
ectool 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>->runProcess(<projectName>, <applicationName>, <processName>,
<tierMapName>, {<optionals>});
```

Example:

```
$ec->runProcess("default", "NewApp", "newProcess", "TierMap2",
{destinationProject => "deploy1"});
```

ectool

Syntax:

```
ectool runProcess <projectName> <applicationName> <processName> <tierMapName>
[optionals...]
```

Example:

```
ectool runProcess default NewApp newProcess TierMap2 --destinationProject
deploy1
```

API Commands - Process Dependency

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.

ec-perl

Syntax:

```
$<object>->createProcessDependency(<projectName>, <processName>,
 <processStepName>, <targetProcessStepName>, {<optionals>});
```

Example:

```
$ec->createProcessDependency("default", "newProcess", "Step C", "Step D",
 {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool createProcessDependency <projectName> <processName> <processStepName>
 <targetProcessStepName> [optionals...]
```

Example:

```
ectool createProcessDependency default newProcess "Step A" "Step "B"
 --componentName VCScomponent
```

deleteProcessDependency

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:

```
$<object>->deleteProcessDependency(<projectName>, <processName>,
 <processStepName>, <targetProcessStepName>, {<optionals>});
```

Example:

```
$ec->deleteProcessDependency("default", "newProcess", "Step B", "Step C",
 {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool deleteProcessDependency <projectName> <processName> <processStepName>
 <targetProcessStepName> [optionals...]
```

Example:

```
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:

```
$<object>->getProcessDependencies(<projectName>, <processName>,
 {<optionals>});
```

Example:

```
$ec->getProcessDependencies("default", "newProcess",
 {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool getProcessDependencies <projectName> <processName> [optionals...]
```

Example:

```
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:

```
$<object>->modifyProcessDependency(<projectName>, <processName>, <processStep
Name>,
<targetProcessStepName>, {<optionals>});
```

Example:

```
$ec->modifyProcessDependency("default", "newProcess", "Step1", "StepA",
 {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool modifyProcessDependency <projectName> <processName> <processStepName>
<targetProcessStepName> [optionals...]
```

Example:

```
ectool modifyProcessDependency default newProcess Step1 StepA --componentName
VCScomponent
```

API Commands - Process Step

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

`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

`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:

```
$<object>->createProcessStep(<projectName>, <processName>,
 <processStepName>, {<optionals>});
```

Example:

```
$ec->createProcessStep("default", "newProcess", "Step 1",
 {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool createProcessStep <projectName> <processName> <processStepName>
 [optionals...]
```

Example:

```
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

`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:

```
$<object>->getProcessStep(<projectName>, <processName>, <processStepName>,
 {<optionals>});
```

Example:

```
$ec->getProcessStep("default", "newProcess", "Step 1",
 {componentName => "VCScomponent"});
```

ectool

Syntax:

```
ectool getProcessStep <projectName> <processName> <processStepName>
 [optionals...]
```

Example:

```
ectool getProcessStep  default newProcess "Step A"
 --componentName VCScomponent
```

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:

```
$<object>->getProcessSteps(<projectName>, <processName>, {<optionals>});
```

Example:

```
$ec->getProcessSteps("default", "newProcess",
 {componentName=> "VCScomponent"});
```

ectool

Syntax:

```
ectool getProcessSteps  <projectName> <processName> [optionals...]
```

Example:

```
ectool getProcessSteps  default newProcess  --componentName VCScomponent
```

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:

```
$<object>->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"
```

API Commands - Project Management

createProject

Creates a new project.

You must specify a `projectName`.

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>   <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

```
$cmdr->createProject("Test Proj", {workspaceName => "Test_WS"});
```

ectool

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

Example

```
ectool createProject "Test Proj" --workspaceName "Test WS"
```

Back to Top

deleteProject

Deletes a project, including all procedures, procedure steps, and jobs within that project.

You must specify a `projectName`.

Arguments	Descriptions
`projectName`	The name of the project you want to delete.

Positional arguments

`projectName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteProject(<projectName>);`

Example

```
$cmdr->deleteProject("Test Proj");
```

ectool

syntax: `ectool deleteProject <projectName>`

Example

```
ectool deleteProject "Test Proj"
```

Back to Top

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"
```

Back to Top

getProjects

Retrieves all projects.

Arguments	Descriptions
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();
```

ectool

syntax: `ectool getProjects`

Example

```
ectool getProjects
```

Back to Top

modifyProject

Modifies an existing project.

You must specify a `projectName`.

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>   <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

```
$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

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.

Arguments	Descriptions
`propertyName`	The name of the property to create. It may be a relative or absolute property path, including "my" paths such as "`/myProject/prop1`".
`applicationName`	The name of the application container of the property sheet which owns the property; must be unique among all projects.
`applicationTierName`	The name of the application tier container of the property sheet which owns the property.
`artifactName`	The name of the artifact container of the property sheet which owns the property.
`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.

Arguments	Descriptions
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.
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
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	Whether or not the property is recursively expandable. <*Boolean flag* - 0|1|true|false> Determines whether the property value will undergo property expansion when it is fetched. Default is "true".
extendedContextSearch	For simple property names, whether or not to search objects in the hierarchy to find the desired property.
gatewayName	The name of the gateway.
groupName	The name of the group where you want to create a 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 the plugin where you want to create a property.

Arguments	Descriptions
procedureName	The name of the procedure. **Must be combined with its 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 container of the property sheet which owns the property; must be unique among all projects.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
propertyType	<string|sheet> Indicates whether to create a string property or a sub-sheet. Default is "string".
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource where you want to define the new property.
resourcePoolName	The name of a pool containing one or more resources.
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	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.
scheduleName	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.
stateDefinitionName	The name of the state definition container of the property sheet which owns the property.
stateName	The name of the state container of the property sheet which owns the property.
stepName	The name of the step container of the property sheet which owns the property.
systemObjectName	The name of the special system object. In this context, only server is legal.
transitionDefinitionName	The name of the transition definition.

Arguments	Descriptions
transitionName	The name of the transition.
userName	The user name where you want to add a property.
value	For a string property (see propertyType above), this is the value of the property. For a sheet property, this argument is invalid.
valueFile	**This option is supported only in Perl and ectool bindings - it is not a part of the XML protocol.** The contents of the *valuefile* is read and stored in the "value" field for a string property. This is an alternative argument for value and is useful if the “value” field spans multiple lines.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace where you want to add a property.
zoneName	The name of the zone.

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

```
$cmdr->createProperty('/myJob/Runtime Env/PATH', {value => 'c:\bin'});

$cmdr->createProperty('Runtime Env/PATH', {value => 'c:\bin', …});
```

ectool

syntax: `ectool createProperty <propertyName> ...`

Examples

```
ectool createProperty "/myJob/Runtime Env/PATH" --value "c:\bin"

ectool createProperty "Runtime Env/PATH" --value "c:\bin" --jobId 4fa765dd-73f1-11e
3-b67e-b0a420524153

ectool createProperty "Saved Variables" --propertyType sheet --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.

Arguments	Descriptions
`propertyName`	The name of the property to delete.
`applicationName`	The name of the application container of the property sheet which owns the property; must be unique among all projects.
`applicationTierName`	The name of the application tier container of the property sheet which owns the property.
`artifactName`	The name of the artifact container of the property sheet which owns the property.
`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`	Whether or not the property is recursively expandable. `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.
`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.

Arguments	Descriptions
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	Descriptions
resourceName	The name of the resource that contains the property you want to delete.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the property you want to delete. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the property you want to delete. **Also requires** projectName and procedureName
systemObjectName	The name of a special system object. Only 'sever' is legal in this context.
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The user name that contains this property.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing this property.
zoneName	The name of the zone.

Positional arguments

propertyName

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteProperty(<propertyName>, { ... });`

Example

```
$cmdr->deleteProperty("/projects/Sample project/Changeset ID");
```

ectool

syntax: `ectool deleteProperty <propertyName> ...`

Example

```
ectool deleteProperty "/projects/Sample project/Changeset ID"
```

Back to Top

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.

Arguments	Descriptions
`value`	The script to evaluate.
`applicationName`	The name of the application container of the property sheet which owns the property; must be unique among all projects.
`applicationTierName`	The name of the application tier container of the property sheet which owns the property.
`artifactName`	The name of the artifact.
`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.
`environmentName`	The name of the environment container of the property sheet which owns the property; must be unique among all projects.

Arguments	Descriptions
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 evaluate a script.
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.
path	Property path string.
pluginName	The name of a plugin where you might evaluate a script.
procedureName	The name of a procedure where you might need to evaluate a script. **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 script to evaluate.
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
stateDefinitionName	The name of the state definition.
stateName	The name of the state.

Arguments	Descriptions
stepName	The name of the step whose script you might evaluate. **Also requires** projectName and procedureName
systemObjectName	System object names include: admin|directory|log|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 where you may need to evaluate a script.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of a workspace where you may need to evaluate a script.
zoneName	The name of the zone.

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-73f1-11
e3-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-b0a420524
153
   --jobStepId 5da765dd-73f1-11e3-b67e-b0a420524153
```

Back to Top

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.

Arguments	Descriptions
`value`	The string value to expand in the given context.
`applicationName`	The name of the application container of the property sheet which owns the property; must be unique among all projects.
`applicationTierName`	The name of the application tier container of the property sheet which owns the property.
`artifactName`	The name of the artifact.
`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.
`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.

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.
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.
path	Property path string.
pluginName	The name of a plugin where you might expand a string.
procedureName	The name of a procedure where you might need to expand a string. **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 string to expand.
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 expand a string.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of a schedule within this project. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step whose string you might be expanding. **Also requires** projectName and procedureName
systemObjectName	System object names include: admin|directory|log|priority|projects|resources|server|session|workspaces

Arguments	Descriptions
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user where you may need to expand the string.
valueFile	**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 "value" field. This is an alternative argument for `value` and is useful if the value field spans multiple lines.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of a workspace where you may need to expand the string.
zoneName	The name of the zone.

Positional arguments

`value`

Response

The expanded string value.

ec-perl

syntax: `$cmdr->expandString(<value>, {<optionals>});`

Examples

```
$cmdr->expandString('$[fullUserName]', {userName => "admin"})->findvalue('//value')
->value();

$cmdr->expandString('$[/myWorkspace/agentUncPath]/$[logFileName]',
   {jobStepId => 5da765dd-73f1-11e3-b67e-b0a420524153})->findvalue('//value')->valu
e();
```

ectool

syntax: `ectool expandString <value> ...`

Examples

```
ectool expandString '$[fullUserName]' --userName admin

ectool expandString '$[/myWorkspace/agentUncPath]/$[logFileName]'
   --jobStepId 5da765dd-73f1-11e3-b67e-b0a420524153
```

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.

Arguments	Descriptions
`applicationName`	The name of the application container of the property sheet which owns the property; must be unique among all projects.
`applicationTierName`	The name of the application tier container of the property sheet which owns the property.
`artifactName`	The name of the artifact container of the property sheet which owns the property.
`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 containing the properties to retrieve. `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`
`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.
`expand`	<*Boolean flag* - `0|1|true|false`> Default value=1 (true), which means the value of each property will be expanded. A value of "0" (false) will cause the unexpanded value of each property to be returned.

Arguments	Descriptions
gatewayName	The name of the gateway.
groupName	The name of the group containing the properties 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`.
path	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.
pluginName	The name of the plugin containing the properties to retrieve.
procedureName	The name of the procedure containing the properties to retrieve. **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 containing the properties to retrieve.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
recurse	<*Boolean flag* - `0|1|true|false`> Default value=0 (false), which means properties of nested sheets will not be included in the response. If you want the properties from all nested sheets to be retrieved, use the value of "1" for true.
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource containing the properties to retrieve.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the properties to retrieve. **Also requires** `projectName`

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->getProperties({<optionals>});`

Examples

```
$cmdr->getProperties({resourceName => "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.

Arguments	Descriptions
`propertyName`	The name or path for the property to retrieve.
`applicationName`	The name of the application container of the property sheet which owns the property; must be unique among all projects.
`applicationTierName`	The name of the application tier container of the property sheet which owns the property.
`artifactName`	The name of the artifact.
`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 containing the property to retrieve. `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`
`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.

Arguments	Descriptions
`expand`	<*Boolean flag* - `0|1|true|false`> Default value=1 (true), which means the value of each property will be expanded. A value of "0" (false) will cause the unexpanded value of each property to be returned.
`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 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`
`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 containing the property to retrieve.
`propertySheetId`	The unique identifier for a property sheet, assigned automatically when the property sheet is created.

Arguments	Descriptions
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource containing the property to retrieve.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the property to retrieve. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the property to retrieve. **Also requires** projectName and procedureName
systemObjectName	The name of the system object containing the property 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 property to retrieve.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing the property to retrieve.
zoneName	The name of the zone.

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"
```

Back to Top

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`.

Arguments	Descriptions
`propertyName`	The name of the property to increment.
`incrementBy`	This is positive or negative integer.
`applicationName`	The name of the application container of the property sheet which owns the property; must be unique among all projects.
`applicationTierName`	The name of the application tier container of the property sheet which owns the property.
`artifactName`	The name of the artifact.
`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.

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.
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.
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 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 "true."
gatewayName	The name of the gateway.
groupName	The name of the group containing the property to increment.
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 a property to increment.
procedureName	The name of the procedure containing this property. **Also requires** projectName

Arguments	Descriptions
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 containing this property. **Also requires** procedureName
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 the resource containing this property.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing this property. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing this property. **Also requires** projectName and procedureName
systemObjectName	Only server is a valid system object for this API.
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user containing this property.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing this property.
zoneName	The name of the zone.

Positional arguments

propertyName, incrementBy

Response

A text string containing the updated numeric property value.

ec-perl

syntax: `$cmdr->incrementProperty(<propertyName> <incrementBy> ...);`

Examples

```
$cmdr->incrementProperty("Build Number", 1, {procedureName => "Delay", projectName
=> "Sample Project");

$cmdr->incrementProperty("/projects/Sample Project/procedures/Delay/Build Number",
1);

$cmdr->incrementProperty("procedures/Delay/Build Number", 1,{projectName => "Sample
Project"});
```

ectool

syntax: `ectool incrementProperty <propertyName> <incrementBy> ...`

Examples

```
ectool incrementProperty "Build Number" 1 --procedureName "Delay" --projectName "Sa
mple Project"

ectool incrementProperty "/projects/Sample Project/procedures/Delay/Build Number" 1

ectool incrementProperty "procedures/Delay/Build Number" 1 --projectName "Sample Pr
oject"
```

Back to Top

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.

Arguments	Descriptions
`propertyName`	The name of the property to be modified; must be unique within the property sheet. This argument can be a path.
`applicationName`	The name of the application container of the property sheet which owns the property; must be unique among all projects.
`applicationTierName`	The name of the application tier container of the property sheet which owns the property.
`artifactName`	The name of the artifact.

Arguments	Descriptions
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.
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	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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
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	Descriptions
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 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 "true."
gatewayName	The name of the gateway.
groupName	The name of the group containing the property to be modified.
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.
newName	Supply any name of your choice to rename the property.
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 be modified.
procedureName	The name of the procedure containing the property to be modified. **Also requires** projectName
projectName	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.
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.
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.

Arguments	Descriptions
propertyType	<string|sheet> Indicates whether to create a string property or a sub-sheet. Default is "string".
repositoryName	The name of the repository for artifact management.
resourceName	The name of the resource containing the property to be modified.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the property to be modified. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the property to be modified. **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 name of the user containing the property to be modified.
value	This can be any string you choose to add to a property.
valueFile	**This option is supported only in Perl and ectool bindings - it is not part of the XML protocol.** The contents of the *valuefile* is read and stored in the "value" field. This is an alternative argument for value and is useful if the "value" field spans multiple lines.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing the property to be modified.
zoneName	The name of the zone.

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
```

Back to Top

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.

Arguments	Descriptions
`propertyName`	The name or path of the property you want to set; must be unique within the property sheet. This argument can be a path.
`value`	The value of the property.
`applicationName`	The name of the application container of the property sheet which owns the property; must be unique among all projects.
`applicationTierName`	The name of the application tier container of the property sheet which owns the property.
`artifactName`	The name of the artifact container of the property sheet which owns the property.

Arguments	Descriptions
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 containing the property you want to set. 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
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
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> Default is "1" (true), which means the property value will be expanded when referenced. If you do not want the property to expand, use the value of "0"(false).

Arguments	Descriptions
extendedContextSearch	<*Boolean flag* - 0|1|true|false> If set, and there is an object specified in the command, ElectricCommander first looks for the property in the object specified, but also searches in other locations if not found, according to the following rules: 1) If the object specified is a procedure, ElectricCommander looks for the property in the project where the procedure resides. 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. Default setting is "false."
gatewayName	The name of the gateway.
groupName	The name of the group containing the property you want to set.
jobId	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.
jobStepId	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.
objectId	This is an object identifier returned by findObjects and getObjects.
notifierName	The name of the email notifier.
pluginName	The name of the plugin containing the property you want to set.
procedureName	The name of the procedure containing the property you want to set. **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 containing the property you want to set.
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	Descriptions
resourceName	The name of the resource containing the property you want to set.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule containing the property you want to set. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step containing the property you want to set. **Also requires** projectName and procedureName
systemObjectName	The name of the system object containing the property you want to set. 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 containing the property you want to set.
valueFile	**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 "value" field. This is an alternative argument for value and is useful if the value field spans multiple lines.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace containing the property you want to set.
zoneName	The name of the zone.

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

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.

Arguments	Descriptions
`resourceNames`	The list of resources to add to the pool.
`resourcePoolName`	The name of a pool containing one or more resources.

Positional arguments

`resourcePoolName, resourceName(s)`

Response

None or status OK message.

ec-perl

syntax:`$cmdr->addResourcesToPool(<resourcePoolName>, {resourceName => [...]});`

Example

```
$cmdr->addResourcesToPool("pool1", { resourceName => ["resource1",
   "resource2", "resource3"]});
```

ectool

syntax:`ectool addResourcesToPool <resourcePoolName> --resourceNames <resourceName1> ...`

(Note the plural form for the resourceNames option)

Example

```
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.

Arguments	Descriptions
`resourceName`	Name for the resource; must be unique among all resources. Argument Type: String
`projectName`	Name for the project; must be unique among all projects; must be unique among all projects. Argument Type: String
`environmentName`	Name of the environment; must be unique among all projects. Argument Type: String
`environmentTierName`	Name for the environment tier; must be unique among all tiers for the environment. Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->addResourceToEnvironmentTier(<resourceName>, <projectName>,
 <environmentName>, <environmentTierName>);
```

Example:

```
$ec->addResourceToEnvironmentTier("Resource1", "default", "newEnv",
 "envTier1");
```

ectool

Syntax:

```
addResourceToEnvironmentTier <resourceName> <projectName> <environmentName>
 <environmentTierName>
```

Example:

```
ectool addResourceToEnvironmentTier Resource1 default newEnv envTier1
```

Back to Top

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`.

Arguments	Descriptions
`artifactCacheDirectory`	The directory on the agent host where retrieved artifacts are stored.
`block`	<*Boolean flag* - `0|1|true|false`> A newly created resource will be pinged. The "block" argument makes the `createResource` call block until the result of the ping is known. Default is "false".
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`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.

Arguments	Descriptions
pools	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.
port	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.
proxyCustomization	Perl code customizing how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources.
proxyHostName	The name or IP address of the computer containing the Commander Agent used for a proxy resource.
proxyPort	The Commander agent port number for a proxy resource. See the port argument description for more details.
proxyProtocol	Protocol for communicating with the proxy target. Defaults to ssh. (This argument is not exposed in the Commander Web Interface at this time.)
resourceDisabled	<*Boolean flag* - 0|1|true|false> If set to 1, Commander will not start new steps on this resource. Defaults to "false".
repositoryNames	A list of one or more repository names—each repository name listed on a "new line".
resourceName	The name of the new resource you are creating.
shell	This sets a default shell for running step commands on this resource. The default is "cmd /q /c" for a Windows agent and "sh -e" for a UNIX agent.
stepLimit	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.

Arguments	Descriptions
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

```
$cmdr->createResource("Test Resource 1", {hostName => "localhost", pools => "P1 P
2"});
```

ectool

syntax: `ectool createResource <resourceName> ...`

Example

```
ectool createResource "Test Resource 1" --hostName localhost --pools "P1 P2"
```

Back to Top

createResourcePool

Creates a new pool for resources.

You must specify a `resourcePoolName`.

Arguments	Descriptions
autoDelete	<*Boolean flag* - 0|1|true|false> - If true, the resource pool will be deleted automatically when the last resource is removed from the pool.
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
resourceNames	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.
orderingFilter	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.
resourcePoolDisabled	<*Boolean flag* - 0|1|true|false> - If true, any runnable steps that refer to the pool will block until the pool is re-enabled.
resourcePoolName	Choose any unique name for your resource pool.

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
```

Back to Top

deleteResource

Deletes a resource.

You must supply a `resourceName`.

Arguments	Descriptions
`resourceName`	The name of the resource to delete.

Positional arguments

`resourceName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteResource(<resourceName>);`

Example

```
$cmdr->deleteResource("Test Resource 1");
```

ectool

syntax: `ectool deleteResource <resourceName>`

Example

```
ectool deleteResource "Test Resource 1"
```

Back to Top

deleteResourcePool

Deletes a resource pool.

You must supply a `resourcePoolName`.

Arguments	Descriptions
`resourcePoolName`	The name of a pool (containing one or more resources) to delete.

Positional arguments

`resourcePoolName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteResourcePool(<resourcePoolName>);`

Example

```
$cmdr->deleteResourcePool("Test Resource 1");
```

ectool

syntax: `ectool deleteResourcePool <resourcePoolName>`

Example

```
ectool deleteResourcePool "Test Resource 1"
```

Back to Top

getResource

Retrieves a resource by its name.

You must specify `resourceName`.

Arguments	Descriptions
`resourceName`	The name of the resource to retrieve.

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"
```

Back to Top

getResources

Retrieves all resources.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more `resource` elements.

ec-perl

syntax: `$cmdr->getResources();`

Example

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

ectool

syntax: `ectool getResources`

Example

```
ectool getResources
```

Back to Top

getResourcesInEnvironmentTier

Returns the list of resources in an environment tier.

You must specify the `projectName`, `environmentName`. and `environmentTierName`arguments.

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects; must be unique among all projects. Argument Type: String
`environmentName`	Name of the environment; must be unique among all projects. Argument Type: String
`environmentTierName`	Name for the environment tier; must be unique among all tiers for the environment. Argument Type: String

Response

Retrieves zero or more resource elements in the specified environment tier.

ec-perl

Syntax:

```
$<object>->getResourcesInEnvironmentTier(<projectName>, <environmentName>,
 <environmentTierName>);
```

Example:

```
$ec->getResourcesInEnvironmentTier("default", "newEnv", "envTier1");
```

ectool

Syntax:

```
getResourcesInEnvironmentTier <projectName> <environmentName>
 <environmentTierName>
```

Example:

```
ectool getResourcesInEnvironmentTier default newEnv envTier1
```

Back to Top

getResourcesInPool

Retrieves a list of resources in a pool.

You must specify a `pool` (name).

Arguments	Descriptions
`jobStepId`	The ID number of the job step related to this pool.
`pool`	The name of a pool containing one or more resources.
`resourcePoolName`	The name of a pool containing one or more resources.

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: `ectool getResourcesInPool <pool>`

Example

```
ectool getResourcesInPool WindowsPool
```

Back to Top

getResourcePool

Retrieves a specified resource pool by name.

You must specify a `resourcePoolName`.

Arguments	Descriptions
`resourcePoolName`	The name of a pool containing one or more resources.

Positional arguments

`resourcePoolName`

Response

An XML stream containing one `resourcePool` element.

ec-perl

syntax: `$cmdr->getResourcePool(<resourcePoolName>);`

Example

```
$cmdr->getResourcePool("WindowsPool");
```

ectool

syntax: `ectool getResourcePool <resourcePoolName>`

Example

```
ectool getResourcePool WindowsPool
```

Back to Top

getResourcePools

Retrieves a list of resource pools.

Arguments	Descriptions
None	

Positional arguments

None

Response

An XML stream containing zero or more `resourcePool`elements.

ec-perl

syntax: `$cmdr->getResourcePools;`

Example

```
$cmdr->getResourcePools;
```

ectool

syntax: `ectool getResourcePools`

Example

```
ectool getResourcePools
```

Back to Top

getResourceUsage

Retrieves resource usage information.

Arguments	Descriptions
None	

Positional arguments

None

Response

An XML stream containing zero or more `resourceUsage` elements.

ec-perl

syntax: `$cmdr->getResourceUsage;`

Example

```
$cmdr->getResourceUsage;
```

ectool

syntax: `ectool getResourceUsage`

Example

```
ectool getResourceUsage
```

Back to Top

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.

Arguments	Descriptions
artifactCacheDirectory	The directory on the agent host where retrieved artifacts are stored.
block	<*Boolean flag* - 0|1|true|false> A newly modified resource will be pinged. The "block" argument makes the modifyResource call "block" until the result of the ping is known. Default is "false".
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
hostName	The name or IP address for the ElectricCommander machine containing the agent for this resource.
newName	Supply any name of your choice to rename the resource.
pools	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.
port	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.
proxyCustomization	Perl code customizing how the proxy resource communicates with the proxy target. Only applicable for proxy resources.
proxyHostName	The IP address of the computer containing the ElectricCommander Agent used for a proxy resource.
proxyPort	The Commander agent port number for a proxy resource. See the port argument for more details.
proxyProtocol	Protocol for communicating with the proxy target. Defaults to ssh. This argument is not exposed in the Commander web interface at this time.
repositoryNames	A list of repository names with each repository name listed on a "new line".
resourceDisabled	<*Boolean flag* - 0|1|true|false> If set to 1, ElectricCommander will not start new steps on this resource.
resourceName	The name of the resource being modified.

Arguments	Descriptions
shell	This sets a default shell for running step commands on this resource. The default is "cmd /q /c" for a Windows agent and "sh -e" for a UNIX agent.
stepLimit	This limits the number of steps that can be running on the resource at one time. Setting this value to "1" is a good way to enforce serial access to the resource.
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 default workspace where job output is stored.
zoneName	The name of the zone where this resource resides.

Positional arguments

resourceName

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->modifyResource(<resourceName>, {...});`

Example

```
$cmdr->modifyResource("Test Resource 1", {stepLimit => 5, shell => "bash"});
```

ectool

syntax: `ectool modifyResource <resourceName> ...`

Example

```
ectool modifyResource "Test Resource 1" --stepLimit 5 --shell "bash"
```

Back to Top

modifyResourcePool

Modifies an existing resource pool.

You must specify a `resourcePoolName`.

Arguments	Descriptions
`autoDelete`	<*Boolean flag* - `0|1|true|false`> - If true, the resource pool will be deleted automatically when the last resource is removed form the pool.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`newName`	Any new unique name you choose to rename this resource pool.
`resourceNames`	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.
`orderingFilter`	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.
`resourcePoolDisabled`	<*Boolean flag* - `0|1|true|false`> - If true, any runnable steps that refer to the pool will block until the pool is re-enabled.
`resourcePoolName`	The name of a pool containing one or more resources.

Positional arguments

`resourcePoolName`

Response

The modified `resourcePool` object.

ec-perl

syntax: `$cmdr->modifyResourcePool(<resourcePoolName>, {<optionals>});`

Example

```
$cmdr->modifyResourcePool("WindowsPool", { resourcePoolDisabled => 1});
```

ectool

syntax: `ectool modifyResourcePool <resourcePoolName> ...`

Example

```
ectool modifyResourcePool WindowsPool --resourcePoolDisabled 1
```

Back to Top

pingAllResources

Pings all resources.

Arguments	Description
`block`	<*Boolean flag* - `0|1|true|false`> Default value="`0`" (false), which means the call will return immediately. If you want the call to wait for responses from every resource before returning, use the value of "`1`"(true).

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
```

Back to Top

pingResource

Pings one resources.

You must specify a `resourceName`.

Arguments	Descriptions
`resourceName`	The name of the resource to ping.

Positional arguments

`resourceName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->pingResource(<resourceName>);`

Example

```
$cmdr->pingResource("Test Resource 1");
```

ectool

syntax: `ectool pingResource <resourceName> ...`

Example

```
ectool pingResource "Test Resource 1"
```

Back to Top

removeResourceFromEnvironmentTier

Removes the given resource from the given environment tier.

You must specify the `resourceName`, `projectName`, `environmentName`. and `environmentTierName`arguments.

Arguments	Descriptions
`resourceName`	Name for the resource; must be unique among all resources. Argument Type: String
`projectName`	Name for the project; must be unique among all projects; must be unique among all projects. Argument Type: String
`environmentName`	Name of the environment; must be unique among all projects. Argument Type: String
`environmentTierName`	Name for the environment tier; must be unique among all tiers for the environment. Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->removeResourceFromEnvironmentTier(<resourceName>, <projectName>,
 <environmentName>, <environmentTierName>);
```

Example:

```
$ec->removeResourceFromEnvironmentTier("Resource"1, "default", "newEnv",
 "envTier1");
```

ectool

Syntax:

```
removeResourceFromEnvironmentTier <resourceName> <projectName>
 <environmentName> <environmentTierName>
```

Example:

```
ectool removeResourceFromEnvironmentTier Resource1 default newEnv envTier1
```

Back to Top

removeResourcesFromPool

Removes resources from a specified resource pool.

You must specify a `resourcePoolName`.

Arguments	Descriptions
`resourceNames`	The list of resources to remove from this pool.
`resourcePoolName`	The name of a pool containing one or more resources.

Positional arguments

`resourcePoolName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->removeResourcesFromPool(<resourcePoolName>, {<optionals>});`

Example

```
$cmdr->removeResourcesFromPool("Test Pool", {resourceNames => ["Test1", "Test2", "T
est3"]});
```

ectool

syntax: `ectool removeResourcesFromPool <resourcePoolName> ...`

Example

```
ectool removeResourcesFromPool "Test Pool" --resourceNames Test1 Test2 Test3
```

Back to Top

API Commands - Schedule Management

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`.

Arguments	Descriptions
`actualParameter`	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.
`beginDate`	<*yyyy-mm-dd*> The date you want the schedule to begin.
`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>   <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.

Arguments	Descriptions
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

```
$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

```
ectool createSchedule "Sample Project" "Weekend" --startTime 00:00
  --stopTime 23:59 --weekDays "Saturday Sunday" --interval 1 --intervalUnits hours
```

Back to Top

deleteSchedule

Deletes a schedule.

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

Arguments	Descriptions
`projectName`	The schedule you want to delete belongs to this project.
`scheduleName`	The name of the schedule you want to delete.

Positional arguments

```
projectName, scheduleName
```

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteSchedule(<projectName>, <scheduleName>);`

Example

```
$cmdr->deleteSchedule("Sample Project","Weekend");
```

ectool

syntax: `ectool deleteSchedule <projectName> <scheduleName>`

Example

```
ectool deleteSchedule "Sample Project" "Weekend"
```

Back to Top

getSchedule

Retrieves a schedule by its name.

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

Arguments	Descriptions
`projectName`	The name of the project that contains the schedule to retrieve.
`scheduleName`	The name of the schedule to retrieve.

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"
```

Back to Top

getSchedules

Retrieves all schedules.

You must specify a `projectName`.

Arguments	Descriptions
projectName	The name of the project containing the schedules to retrieve.

Positional arguments

projectName

Response

Zero or more schedule elements for all schedules within the named project.

ec-perl

syntax: `$cmdr->getSchedules(<projectName >);`

Example

```
$cmdr->getSchedules("Sample Project");
```

ectool

syntax: `ectool getSchedules <projectName>`

Example

```
ectool getSchedules "Sample Project"
```

Back to Top

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.

Arguments	Descriptions
actualParameter	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.
beginDate	<*yyyy-mm-dd*> The date you want the schedule to begin.
clearActualParameters	<*Boolean flag* - 0|1|true|false> If set to true, all actual parameters will be removed from the schedule.

Arguments	Descriptions
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
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	Descriptions
projectName	The name of the project containing the schedule to modify.
scheduleDisabled	<*Boolean flag* - 0|1|true|false> If set to 1, Commander will not start any new jobs from the schedule.
scheduleName	The name of the schedule to modify.
startTime	Enter hours and minutes, formatted hh:mm, using the 24-hour clock. Commander starts creating jobs at this time on the days specified.
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, Commander creates one job only on each specified day.
timeZone	Supply the time zone 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 a status OK message.

ec-perl

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

Example

```
$cmdr->modifySchedule("Sample Project", "Weekend",
      {procedureName => "Delay",
     actualParameter => {actualParameterName => "Delay Time",
                                       value => "5"}});
```

ectool

syntax: `ectool modifySchedule <projectName> <scheduleName> ...`

Example

```
ectool modifySchedule "Sample Project" "Weekend" --procedureName "Delay"
   --actualParameter "Delay Time=5"
```

Back to Top

API Commands - Server Management

getVersions

Retrieves server version information.

Arguments	Descriptions
None	

Positional arguments

None

Response

A serverVersion element.

ec-perl

syntax: `$cmdr->getVersions();`

Example

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

ectool

syntax: `ectool getVersions`

Example

```
ectool getVersions
```

Back to Top

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	Descriptions
`force`	<*Boolean flag* - `0|1|true|false`> The "1" flag tells the Commander server to exit immediately, without performing any of the usual associated cleanup activities. This action "kills" all running jobs.
`restart`	<*Boolean flag* - `0|1|true|false`> The "1" flag tells the Commander server to shut down normally and immediately start again.

Positional arguments

None

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->shutdownServer({restart => 1});
```

ectool

syntax: `ectool shutdownServer ...`

Example

```
ectool shutdownServer --restart 1
```

Back to Top

importLicenseData

Imports one or more licenses.

You must specify `licenseData`.

Arguments	Descriptions
`licenseData`	The content of a license file (`perl|XML API`).
`licenseFile`	<*localFileName*> The license fie to import. This is a local file that will be read by ectool. The contents is sent as the `licenseData` argument (ectool only).

Positional arguments

`licenseData`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->importLicenseData(<licenseData>)`

Example

```
my $data = 'cat license.xml';
 $cmdr->importLicenseData ($data);
```

ectool

syntax: `ectool importLicenseData <licenseData>`

Example

```
ectool importLicenseData license.xml
```

Back to Top

getAdminLicense

Retrieves the admin license, which can be used when all concurrent user licenses are in use.

Arguments	Descriptions
None	

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:

```
<error requestId="1">
    <code>AccessDenied</code>
    <where></where>
    <message>Principal 'bob@company.com' does not have execute privileges on
          systemObject[name=licensing,id=10]</message>
    <details></details>
</error>
```

When the user has permission to get/use the Administrator license, but already has a User license:

```
 <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>
```

ec-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`.

Arguments	Descriptions
`featureName`	The name of the licensed feature. Possible features include: `Server`
`productName`	The name of the product with the licensed feature. Possible products include: `ElectricCommander`

Positional arguments

`productName, featureName`

Response

One `license` element.

ec-perl

syntax: `$cmdr->getLicense(<productName>, <featureName>);`

Example

```
$cmdr->getLicense('ElectricCommander', 'Server');
```

ectool

syntax: `ectool getLicense <productName> <featureName>`

Example

```
ectool getLicense ElectricCommander Server
```

Back to Top

getLicenses

Retrieves all license data.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more `license` elements.

ec-perl

syntax: `$cmdr->getLicenses();`

Example

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

ectool

syntax: `ectool getLicenses`

Example

```
ectool getLicenses
```

Back to Top

getLicenseUsage

Retrieves the current license usage.

Arguments	Descriptions
None	

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
```

Back to Top

deleteLicense

Deletes a license.

You must specify a `productName` and `featureName`.

Arguments	Descriptions
`featureName`	The name of the licensed feature. Possible features include: `Server`
`productName`	The name of the product with the licensed feature. Possible products include: `ElectricCommander`

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
```

Back to Top

getServerStatus

Retrieves the current status of the ElectricCommander server.

Arguments	Descriptions
`block`	<*Boolean flag* - `0|1|true|false`> A "1" flag causes the API to wait until the server reaches a "terminal state". Terminal states include `running`, `failed`, `stopping`, and `importFailed`.
`diagnostics`	<*Boolean flag* - `0|1|true|false`> This argument supplies the following diagnostics" information in your output: • `threadDump` - stack dumps of all threads in the server • `statistics` - output from all system timers • `systemProperties` - values of all java system properties • `environmentVariables` - values of all environment variables • `settings` - values of all server settings • `serverInfo` - output from `getServerInfo` call.
`serverStateOnly`	<*Boolean flag* - `0|1|true|false`> A "1" flag causes the API to limit the response to the short form, and causes ectool to return only the value of the `serverStatus` element as a simple string value.
`timeout`	This flag specifies the timeout for the `element` flag. The default value is 120 seconds.

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

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:

```
$<object>->createTierMap(<projectName>, <applicationName>,
 <environmentProjectName>, <environmentName>), {<optionals>});
```

Example:

```
$ec->createTierMap("default", "newApp", "defaultEnv", "Env1",
{tierMapping => [{applicationTier => "AppTier1",
environmentTier => "EnvTier1"}, {applicationTier => "AppTier2",
environmentTier => "EnvTier2"}], tierMapName => "TierMap1"});
```

ectool

Syntax:

```
ectool createTierMap <projectName> <applicationName>
 <environmentProjectName> <environmentName> [optionals...]
```

Example:

```
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:

```
$<object>->deleteTierMap(<projectName>, <applicationName>,
 <environmentProjectName>, <environmentName>);
```

Example:

```
$ec->deleteTierMap("default", "App1", "MyProj", "Env1");
```

ectool

Syntax:

```
ectool deleteTierMap <projectName> <applicationName>
 <environmentProjectName> <environmentName>
```

Example:

```
ectool deleteTierMap default TierMapToDelete defaultEnv Env1
```

deleteTierMapping

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.

ec-perl

Syntax:

```
$<object>->deleteTierMapping(<projectName>, <applicationName>,
 <environmentProjectName>, <environmentName>, <applicationTierName>);
```

Example:

```
$ec->deleteTierMap("default", "App1", "MyProj", "Env1",
"InstallTier");
```

ectool

Syntax:

```
ectool deleteTierMapping <projectName> <applicationName>
 <environmentProjectName> <environmentName> <applicationTierName>
```

Example:

```
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:

```
$<object>->getTierMaps(<projectName>, <applicationtName>);
```

Example:

```
$ec->getTierMaps("default", "NewApp");
```

ectool

Syntax:

```
ectool getTierMaps <projectName> <applicationtName>
```

Example:

```
ectool getTierMaps default NewApp
```

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:

```
$<object>->modifyTierMap(<projectName>, <applicationName>,
 <environmentProjectName>, <environmentName>), {<optionals>});
```

Example:

```
$ec->modifyTierMap("default", "newApp", "defaultEnv", "Env1",
{tierMapping => [{applicationTier => "AppTier1",
environmentTier => "EnvTier1"}, {applicationTier => "AppTier2",
environmentTier => "EnvTier2"}], tierMapName => "TierMap1"});
```

ectool

Syntax:

```
ectool modifyTierMap <projectName> <applicationName>
 <environmentProjectName> <environmentName> [optionals...]
```

Example:

```
ectool modifyTierMap default newApp defaultEnv Env1 --tierMapName TierMap1
--tierMapping AppTier1=EnvTier1 AppTier2=EnvTier2
```

API Commands - User/Group Management

addUsersToGroup

Adds ones or more specified users to a particular group.

You must specify a `groupName` and one or more user names.

Arguments	Descriptions
`groupName`	The name of the group you are modifying.
`userNames`	The list of user names to add to this group.

Positional arguments

`groupName, userNames`

Response

None or status OK message.

ec-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
```

Back to Top

createGroup

Creates a new local group of users.

You must specify a `groupName`.

Arguments	Descriptions
`groupName`	A name you choose for the new group you are creating.
`userNames`	One or more user names to add to the group.

Positional arguments

`groupName`

Response

None or status OK message.

ec-perl

syntax: `$cmdr->createGroup(<groupName>, {<optionals>});`

Example

```
$cmdr->createGroup("Build Users", {userName => ["aallen", "Betty Barker", "cclar
k"]});
```

ectool

syntax: `ectool createGroup <groupName> --userNames <user1> ...`
(Note the plural form of `userNames`.)

Example

```
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:

```
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:

```
# 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`.

Arguments	Descriptions
`email`	The new user's email address.
`fullUserName`	The user's full name - not his or her nickname.
`groupNames`	*<group1 group2>* Any group name containing spaces must be enclosed in double-quotes.
`password`	The new user's password.
`userName`	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.

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`.

Arguments	Descriptions
`groupName`	The name of the group you want to delete.

Positional arguments

`groupName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteGroup(<groupName>);`

Example

```
$cmdr->deleteGroup("Build Users");
```

ectool

syntax: `ectool deleteGroup <groupName>`

Example

```
ectool deleteGroup "Build Users"
```

Back to Top

deleteUser

Deletes a local user.

You must specify the `userName`.

Arguments	Descriptions
`userName`	The name of the user you want to delete.

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"
```

Back to Top

getGroup

Retrieves a group by its name.

You must specify the `groupName`.

Arguments	Descriptions
`groupName`	The name of the group to retrieve.
`providerName`	Using this option allows you to search only the specified provider for group information. (LDAP or Active Directory)

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
```

Back to Top

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.

ec-perl

syntax: $cmdr->getUser(<userName>, {<optionals>});

Example

```
$cmdr->getUser("Betty Barker");
```

ectool

syntax: ectool getUser <userName> ...

Example

```
ectool getUser "Betty Barker"
```

Back to Top

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.

Arguments	Descriptions
filter	<*filter pattern*> Supply a filter pattern to match user names. The filter is not case sensitive and can include the "*" wildcard character.
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. 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	<*number of users*> Specify a larger number of user records to retrieve. The default limit is 100 user records.

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`.

Arguments	Descriptions
`password`	The password for the user who is "logging in".
`userName`	The name of a user who has login privileges.

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"
```

Back to Top

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
```

Back to Top

modifyGroup

Modifies an existing group.

You must specify `groupName`.

Arguments	Descriptions
`groupName`	The name of the group to modify.
`migrateSettings`	<*targetGroupName*> Use this argument to specify the new name to which the settings need to be moved.
`newName`	Supply any name of your choice to rename the group.
`removeAllUsers`	<*Boolean flag* - `0|1|true|false`>

Arguments	Descriptions
userNames	user1 [user2...] 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
```

Back to Top

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:

```
# 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`.

Arguments	Descriptions
`email`	The user's email address.
`fullUserName`	The user's full name. For example, "John Smith".
`groupNames`	*group1* [*group2* ...] Assigns the user to one or more groups and removes the user from any groups not included in the list.
`migrateSettings`	<*targetUserName*> Use this option to specify the new name to which the settings need to be moved.
`newName`	The user's new name (for example, if changing an existing user's surname).
`password`	Supply a new password to set for the user.
`removeFromAllGroups`	<*Boolean flag* - `0|1|true|false`> If set to 1, this user will be removed from all groups.
`sessionPassword`	If changing the user's password, you must supply the password used in the "login" command also.
`userName`	The name used by the user to login and/or receive email. For example, "jsmith".

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.

Arguments	Descriptions
`groupName`	The name of the group from which to remove users.
`userNames`	The list of users to remove from the group.

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`.

Arguments	Descriptions
projectName	The name of the project.
workflowName	The name of the workflow.

Positional arguments

`projectName, workflowName`

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->completeWorkflow ("projectA", "workflow_26_201010121647");
```

ectool

syntax: `ectool completeWorkflow <projectName> <workflowName>`

Example

```
ectool completeWorkflow projectA workflow_26_201010121647
```

Back to Top

deleteWorkflow

Deletes a workflow, including all states and transitions.

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

Arguments	Descriptions
`deleteProcesses`	*<Boolean flag -* `0|1|true|false`*>*
`projectName`	The name of the project containing the workflow to delete.
`workflowName`	The name of the workflow.

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
```

Back to Top

getState

Finds a state by name.

You must specify `projectName`, `workflowName`, and `stateName`.

Arguments	Descriptions
`projectName`	The name of the project containing the state.
`stateName`	The name of the state.
`workflowName`	The name of the workflow.

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
```

Back to Top

getStates

Retrieves all states in a workflow.

You must specify `projectName` and `workflowName`.

Arguments	Descriptions
`projectName`	The name of the project containing the state.
`workflowName`	The name of the workflow.

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
```

Back to Top

getTransition

Finds a transition by name.

You must specify `projectName`, `workflowName`, `stateName`, and `transitionName`.

Arguments	Descriptions
`projectName`	The name of the project containing the transition.
`stateName`	The name of the state.
`transitionName`	The name of the transition.
`workflowName`	The name of the workflow.

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`.

Arguments	Descriptions
`projectName`	The name of the project containing the transition.

Arguments	Descriptions
stateName	The name of the state.
targetState	The target state for the transition definition.
workflowName	The name of the workflow.

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
```

Back to Top

getWorkflow

Finds a workflow by name.

You must specify a projectName and workflowName.

Arguments	Descriptions
projectName	The name of the project containing the workflow.
workflowName	The name of the workflow.

Positional arguments

projectName, workflowName

Response

One workflow element.

ec-perl

syntax: `$cmdr->getWorkflow (<projectName>, <workflowName>);`

Example

```
$cmdr->getWorkflow ("projectA", "BTD");
```

ectool

syntax: `ectool getWorkflow <projectName> <workflowName>`

Example

```
ectool getWorkflow projectA BTD
```

Back to Top

getWorkflows

Retrieves all workflow instances in a project.

You must specify a `projectName`.

Arguments	Descriptions
`projectName`	The name of the project containing the workflows.

Positional arguments

`projectName`

Response

Zero or more `workflow` elements.

ec-perl

syntax: `$cmdr->getWorkflows (<projectName>);`

Example

```
$cmdr->getWorkflows ("projectA");
```

ectool

syntax: `ectool getWorkflows <projectName>`

Example

```
ectool getWorkflows projectA
```

Back to Top

runWorkflow

Runs the specified workflow definition and returns the workflow name.

You must specify the `projectName` and `workflowDefinitionName`.

Arguments	Descriptions
`actualParameter`	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.
`projectName`	The name of the project containing the workflow definition.
`startingState`	The initial state of the workflow.
`workflowDefinitionName`	The name of the workflow definition.

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
```

Back to Top

transitionWorkflow

Manually transition from the active workflow state.

You must specify `projectName`, `workflowName`, `stateName`, and `transitionName`.

Arguments	Descriptions
`actualParameter`	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.

Arguments	Descriptions
projectName	The name of the project containing the workflow to transition.
stateName	The name of the state.
transitionName	The name of the transition.
workflowName	The name of the workflow to transition.

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

API Commands - Workflow Definition Management

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`.

Arguments	Descriptions
`actualParameter`	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.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`projectName`	The name of the project.
`startable`	<*Boolean flag* - `0|1|true|false`> "True" means this state definition can be the initial state of an instantiated workflow.
`stateDefinitionName`	Choose any unique name of your choice for the state definition. This name must be unique within the workflow definition.
`subprocedure`	Name of the procedure launched when the state is entered. **Also requires** `subproject`
`subproject`	Name of the project containing the procedure or workflow launched when the state is entered.

Arguments	Descriptions
substartingState	Name of the starting state for the workflow launched when the state is entered. **Also requires** subproject and subworkflowDefinition
subworkflowDefinition	Name of the workflow definition launched when the state is entered. **Also requires** subproject
workflowDefinitionName	The name of the workflow definition.

Positional arguments

projectName, workflowDefinitionName, stateDefinitionName

Response

One stateDefinition element.

ec-perl

syntax: `$cmdr->createStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, {<optionals>});`

Example

```
$cmdr->createStateDefinition ("ProjectA", "BTD", "build", {startable => 1,
      subproject => "product",
    subprocedure => "Master",
     description => "free text"});
```

ectool

syntax: `ectool createStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> ...`

Example

```
ectool createStateDefinition ProjectA BTD build  --startable 1 --subproject product
    --subprocedure Master --description "free text"
```

Back to Top

createTransitionDefinition

Creates a new transition definition for workflow definition.

You must specify projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName, and targetState.

Arguments	Descriptions
actualParameter	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.
condition	A fixed text or text embedding property references that are evaluated into a logical TRUE or FALSE. An empty string, a "0" or "false" 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.
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
projectName	The name of the project.
stateDefinitionName	The name of the state definition.
targetState	Target state for the transition definition.
transitionDefinitionName	Choose any unique name of your choice for the transition definition. This name must be unique within the state definition.
trigger	Possible values are: onEnter|onStart|onCompletion|manual
workflowDefinitionName	The name of the workflow definition.

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", "tes
t",
    {trigger => "manual", description => "free text"});
```

ectool

syntax: `ectool createTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName> <targetState> ...`

Example

```
ectool createTransitionDefinition ProjectA BTD build build2test test --trigger manu
al
   --description "free text"
```

Back to Top

createWorkflowDefinition

Creates a new workflow definition for a project.

You must supply a `projectName` and a `workflowDefinitionName`.

Arguments	Descriptions
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`projectName`	The name of the project containing the workflow.
`workflowDefinitionName`	Choose any unique name of your choice for the workflow definition. This name must be unique within the project.
`workflowNameTemplate`	The name of the workflow template.

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> ...`

Example

```
ectool createWorkflowDefinition projectA BTD --description "free text"
```

Back to Top

deleteStateDefinition

Deletes a state definition.

You must specify a `projectName`, `workflowDefinitionName`, and `stateDefinitionName`.

Arguments	Descriptions
`projectName`	The name of the project containing the state definition.
`workflowDefinitionName`	The name of the workflow definition.
`stateDefinitionName`	The name of the state definition.

Positional arguments

```
projectName, workflowDefinitionName, stateDefinitionName
```

Response

None or status OK message.

ec-perl

syntax: `$cmdr->deleteStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>);`

Example

```
$cmdr->deleteStateDefinition ("projectA", "BTD", "build");
```

ectool

syntax: `ectool deleteStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName>`

Example

```
ectool deleteStateDefinition projectA BTD build
```

Back to Top

deleteTransitionDefinition

Deletes a transition definition.

You must specify a `projectName`, `workflowDefinitionName`, `stateDefinitionName`, and `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

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
```

Back to Top

deleteWorkflowDefinition

Deletes a workflow definition, including all state and transition definitions.

You must specify a `projectName` and a `workflowDefinitionName`

Arguments	Descriptions
projectName	The name of the project containing the workflow definition to delete.
workflowDefinitionName	The name of the workflow definition.

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
```

Back to Top

getStateDefinition

Finds a state definition by name.

You must specify `projectName`, `workflowDefinitionName`, and `stateDefinitionName`.

Arguments	Descriptions
`projectName`	The name of the project containing the state definition.
`stateDefinitionName`	The name of the state definition.
`workflowDefinitionName`	The name of the workflow definition.

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
```

Back to Top

getStateDefinitions

Retrieves all state definitions in a workflow definition.

You must specify `projectName` and `workflowDefinitionName`.

Arguments	Descriptions
`includeFormalParameters`	<*Boolean flag* - `0|1|true|false`>
`projectName`	The name of the project containing the state definition.
`startableOnly`	<*Boolean flag* - `0|1|true|false`>
`workflowDefinitionName`	The name of the workflow definition.

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
```

Back to Top

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
```

Back to Top

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.

ec-perl

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
```

Back to Top

getWorkflowDefinition

Finds a workflow definition by name.

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

Arguments	Descriptions
`projectName`	The name of the project containing the workflow definition.
`workflowDefinitionName`	The name of the workflow definition.

Positional arguments

```
projectName, workflowDefinitionName
```

Response

One `workflowDefinition` element.

ec-perl

syntax: `$cmdr->getWorkflowDefinition (<projectName>, <workflowDefinitionName>);`

Example

```
$cmdr->getWorkflowDefinition ("projectA", "BTD");
```

ectool

syntax: `ectool getWorkflowDefinition <projectName> <workflowDefinitionName>`

Example

```
ectool getWorkflowDefinition projectA BTD
```

Back to Top

getWorkflowDefinitions

Retrieves all workflow definitions in a project.

You must specify a `projectName`.

Arguments	Descriptions
`projectName`	The name of the project containing the workflow definitions.

Positional arguments

`projectName`

Response

Zero or more `workflowDefinition` elements.

ec-perl

syntax: `$cmdr->getWorkflowDefinitions (<projectName>);`

Example

```
$cmdr->getWorkflowDefinitions ("projectA");
```

ectool

syntax: `ectool getWorkflowDefinitions <projectName>`

Example

```
ectool getWorkflowDefinitions projectA
```

Back to Top

modifyStateDefinition

Modifies an existing state definition.

You must specify `projectName`, `workflowDefinitionName`, and `stateDefinitionName`.

Arguments	Descriptions
`actualParameter`	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.

Arguments	Descriptions
clearActualParameters	<*Boolean flag* - 0|1|true|false>
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
newName	The new name of your choice for the state definition.
projectName	The name of the project containing the state definition.
startable	<*Boolean flag* - 0|1|true|false>
stateDefinitionName	The name of the state definition to modify.
subprocedure	The name of the procedure launched when the state is entered. **Also requires** subproject
subproject	The name of the project containing the procedure or workflow launched when the state is entered.
substartingState	The name of the workflow starting state that is launched when the state is entered. **Also requires** subproject and subworkflowDefinition
subworkflowDefinition	The name of the workflow definition launched when the state is entered. **Also requires** subproject
workflowDefinitionName	The name of the workflow definition.

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",
```

```
        subprocedure => "Master",
         description => "sample text"});
```

ectool

syntax: `ectool modifyStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> ...`

Example

```
ectool modifyStateDefinition projectA BTD build --startable 1 --subproject factory
   --subprocedure Master --description "sample text"
```

Back to Top

modifyTransitionDefinition

Modifies an existing transition definition.

You must specify `projectName`, `workflowDefinitionName`, `stateDefinitionName`, and `transitionDefinitionName`.

Arguments	Descriptions
`actualParameter`	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.
`clearActualParameters`	<*Boolean flag* - `0|1|true|false`>
`condition`	A fixed text or text embedded property references that are evaluated into a logical "true" or "false". An empty string, a "0" or "false" 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.
`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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`newName`	A new name of your choice for the transition definition--must be a unique name within the workflow.
`projectName`	The name of the project containing the transition definition to modify.
`stateDefinitionName`	The name of the state definition.

Arguments	Descriptions
targetState	The target state for the transition definition.
transitionDefinitionName	The name of the transition definition to modify.
trigger	Possible values are: onEnter|onStart|onCompletion|manual
workflowDefinitionName	The name of the workflow definition.

Positional arguments

```
projectName, workflowDefinitionName, stateDefinitionName,
transitionDefinitionName
```

Response

One transitionDefinition element.

ec-perl

syntax: `$cmdr->modifyTransitionDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, <transitionDefinitionName>, {<optionals>});`

Example

```
$cmdr->modifyTransitionDefinition ("projectA", "BTD", "build", "build2test",
        {targetState => "deploy",
             trigger => "onCompletion",
         description => "bypass all tests"});
```

ectool

syntax: `ectool modifyTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName> ...`

Example

```
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.

Arguments	Descriptions
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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
newName	The new name of your choice for the workflow definition--must be a unique name within the workflow.
projectName	The name of the project containing the workflow definition to modify.
workflowDefinitionName	The name of the workflow definition to modify.
workflowNameTemplate	The template used to determine default names for workflows launched from a workflow definition.

Positional arguments

projectName, workflowDefinitionName

Response

One workflowDefinition element.

ec-perl

syntax: $cmdr->modifyWorkflowDefinition (<projectName>, <workflowDefinitionName>, {<optionals>});

Example

```
$cmdr->modifyWorkflowDefinition ("projectA", "BTD",
        {newName => "BuildTestDeploy",
     description => "changed name"});
```

ectool

syntax: ectool modifyWorkflowDefinition <projectName> <workflowDefinitionName> ...

Example

```
ectool modifyWorkflowDefinition projectA BTD
   --newName "BuildTestDeploy"
   --description "changed name"
```

Back to Top

moveStateDefinition

Moves a state definition within a workflow definition.

You must specify projectName, workflowDefinitionName, and stateDefinitionName.

Arguments	Descriptions
beforeStateDefinition	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.
projectName	The name of the project containing the state definition.
stateDefinitionName	The name of the state definition to move.
workflowDefinitionName	The name of the workflow definition.

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
```

Back to Top

moveTransitionDefinition

Moves a transition definition within a workflow definition.

You must specify projectName, workflowDefinitionName, stateDefinitionName, and transitionDefinitionName.

Arguments	Descriptions
beforeTransitionDefinition	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.
projectName	The name of the project containing the transition definition.
stateDefinitionName	The name of the state definition.
transitionDefinitionName	The name of the transition definition to move.
workflowDefinitionName	The name of the workflow definition.

Positional arguments

```
projectName, workflowDefinitionName, stateDefinitionName,
transitionDefinitionName
```

Response

None or status OK message.

ec-perl

syntax: `$cmdr->moveTransitionDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, <transitionDefinitionName>, {<optionals>});`

Example

```
$cmdr->moveTransitionDefinition ("projectA", "BTD", "Build", "in",
    {beforeTransitionDefinition => "out"});
```

ectool

syntax: `ectool moveTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName> ...`

Example

```
ectool moveTransitionDefinition projectA BTD Build in--beforeTransitionDefinition o
ut
```

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`**:

agentDrivePath	agentUncPath	Result from running a step in "job123" that uses this workspace
`N:\`	`\\server\share`	The agent maps `\\server\share to drive n:` and runs the step in `n:\job123.`
`N:\sub1`	`\\server\share\dir1\sub1`	The agent maps `\\server\share\dir1` to drive `n:` and runs the step in `n:\sub1\job123.`
`N:\sub1`	`\\server\share\dir1`	Invalid! No mapping can be deduced from this pair of values.
`C:\ws`	`C:\ws`	A local workspace on the agent. No drive mapping is needed. The job step runs in `c:\ws\job123.`
`C:\ws`		Same as if `agentUncPath` were set identical to `agentDrivePath.`

You must specify a `workspaceName`.

Arguments	Descriptions
`agentDrivePath`	Drive-letter-based path used by Windows agents to access the workspace in steps.

Arguments	Descriptions
agentUncPath	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.
agentUnixPath	UNIX path used by UNIX agents and Linux Commander Web servers to access the workspace.
credentialName	Credential to use when connecting to a network location. 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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
local	<*Boolean flag* - 0|1|true|false> Set to "true", the workspace is local.
workspaceDisabled	<*Boolean flag* - 0|1|true|false> Set to "true", the workspace is disabled.
workspaceName	Any name you choose to name this workspace.
zoneName	The name of the zone where this workspace resides.

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'
```

Back to Top

deleteWorkspace

Deletes a workspace.

You must specify the `workspaceName`.

Arguments	Descriptions
`workspaceName`	The name of the workspace to delete.

Positional arguments

`workspaceName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->deleteWorkspace(<workspaceName>);`

Example

```
$cmdr->deleteWorkspace("test");
```

ectool

syntax: `ectool deleteWorkspace <workspaceName>`

Example

```
ectool deleteWorkspace test
```

Back to Top

getWorkspace

Retrieves a workspace by name.

You must specify the `workspaceName`.

Arguments	Descriptions
`workspaceName`	The name of the workspace to retrieve.

Positional arguments

workspaceName

Response

One workspace element.

ec-perl

syntax: $cmdr->getWorkspace(<workspaceName>);

Example

```
$cmdr->getWorkspace("test");
```

ectool

syntax: ectool getWorkspace <workspaceName>

Example

```
ectool getWorkspace test
```

Back to Top

getWorkspaces

Retrieves all workspaces.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more workspace elements.

ec-perl

syntax: $cmdr->getWorkspaces();

Example

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

ectool

syntax: ectool getWorkspaces

Example

```
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`:

agentDrivePath	agentUncPath	Result from running a step in "job123" that uses this workspace
`N:\`	`\\server\share`	The agent maps `\\server\share` to drive `n:` and runs the step in `n:\job123`.
`N:\sub1`	`\\server\share\dir1\sub1`	The agent maps `\\server\share\dir1` to drive `n:` and runs the step in `n:\sub1\job123`.
`N:\sub1`	`\\server\share\dir1`	Invalid! No mapping can be deduced from this pair of values.
`C:\ws`	`C:\ws`	A local workspace on the agent. No drive mapping is needed. The job step runs in `c:\ws\job123`.
`C:\ws`		Same as if `agentUncPath` were set identical to `agentDrivePath`.

You must specify a `workspaceName`.

Arguments	Descriptions
`agentDrivePath`	Drive-letter-based path used by Windows agents to access the workspace in steps.
`agentUncPath`	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.
`agentUnixPath`	UNIX path used by UNIX agents and Linux Commander web servers to access the workspace.

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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
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.

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. 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.
configName	The name of the email configuration.
groupName	The full name of a group. For Active Directory and LDAP, this is a full DN.
newOwnerName	The name of the new owner for this object. Defaults to the current user.
notifierName	The name of the email notifier.
pluginName	The name of the plugin - the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.
procedureName	The name of the procedure - may be a path to the procedure. **Also requires** projectName

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.

Arguments	Descriptions
Naming	
cloneName	The cloneName 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.
Locators	
applicationName	The name of the application container of the property sheet which owns the property; must be unique among all projects.
applicationTierName	The name of the application tier container of the property sheet which owns the property.
artifactName	The name of the artifact container of the property sheet which owns the property.
artifactVersionName	The name of the artifactVersion container of the property sheet which owns the property..
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.

Arguments	Descriptions
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.
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 which owns the 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	The object id as returned by FindObjects.
path	The property path that specifies the object.
pluginName	The name of the plugin container of the property sheet which owns the property.
procedureName	The name of the procedure you want to clone. **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 you want to clone.

Arguments	Descriptions
propertySheetId	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
providerName	The LDAP or Active Directory provider name.
resourceName	The name of the resource you want to clone.
resourcePoolName	The name of a pool containing one or more resources.
scheduleName	The name of the schedule you want to clone. **Also requires** projectName
stateDefinitionName	The name of the state definition.
stateName	The name of the state.
stepName	The name of the step you want to clone. **Also requires** projectName and procedureName
systemObjectName	System object names include: admin|artifacts|directory|emailConfigs|forceAbort|licensing|log| plugins|priority|projects|repositories|resources|server|session| workspaces|zonesAndGateways
transitionDefinitionName	The name of the transition definition.
transitionName	The name of the transition.
userName	The name of the user where you may need to expand the string.
workflowDefinitionName	The name of the workflow definition.
workflowName	The name of the workflow.
workspaceName	The name of the workspace you want to clone.
zoneName	The name of the zone.

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.

$xPath = $cmdr->clone(
    {
        projectName => "EC-Examples",
        procedureName => "set Property"
    }
);


# Create a copy of a procedure providing a name for the copy.

$xPath = $cmdr->clone(
    {
        projectName   => "EC-Examples",
        procedureName => "set Property",
        cloneName     => "set Property 2"
    }
);


# Create a copy of a procedure step.

$xPath = $cmdr->clone(
    {
        projectName   => "EC-Examples",
        procedureName => "set Property",
        cloneName     => "set Property 2",
        stepName      => 'setProperty'
    }
);

# Copy a step using the path.

$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/setPrope
rty'
<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.

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 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.
	
	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	Descriptions
objectType	This argument specifies the type of object to count. Values include: artifact project artifactVersion property credential repository directoryProvider resource emailconfig resourcePool emailNotifier schedule formalParameter state job stateDefinition jobStep transition logEntry transitionDefinition plugin workflow procedure workflowDefinition procedureStep workspace

Positional arguments

objectType

Response

Returns the number of filtered objects.

ec-perl

syntax: `$cmdr->countObjects(<objectType>, {<optionals>});`

Example

```
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
arbinary date
                    "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`.

Arguments	Descriptions

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	Descriptions
`objectType`	This argument specifies the type of object to find. Values include: `artifact|artifactVersion|job|logEntry|project|repository|workflow`
`sorts`	Specify "sorts" 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.

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.

```
 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.

Back to Top

export

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.

Arguments	Descriptions
`compress`	<*Boolean flag* - `0|1|true|false`> Use this argument to compress XML output. If set to 1, the file will be compressed using the “gzip” format and a “.gz” file extension will be added to the filename. The default behavior is to compress the output. **Note:** This is true for full exports only, not a partial export.
`excludeJobs`	<*Boolean flag* - `0|1|true|false`> If set to 1, no job information will be exported. This argument can be used to reduce the size of the export file.
`fileName`	<*remoteFileName*> 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.
`path`	<*property path*> 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.
`relocatable`	<*Boolean flag* - `0|1|true|false`> If the `--relocatable` flag is set to "true", a partial export (for example, with `--path`) will not include object IDs, ACLs, system property sheets, create/modify times, owners, email notifiers or `lastModifiedBy` information, and the export file result will be much smaller than a normal export. When this file is imported, the result should show one or more objects owned by the importing user as if they were newly created. **Note:** The relocatable argument ONLY works with a partial export. This argument is silently ignored during a full export.

Arguments	Descriptions
safeMode	The safeMode 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: • none (default) - Do not quiesce the server during export. • shutdown - Quiesce the server and shutdown when complete. • restart - Quiesce the server and restart when complete. **Note:** The safeMode argument has no effect on partial exports.
withAcls	**Modifies** relocatable. <*Boolean flag* - 0|1|true|false> If the withAcls flag is set to "true", a relocatable partial export will include ACLs.
withNotifiers	**Modifies** relocatable. <*Boolean flag* - 0|1|true|false> If the withNotifiers flag is set to "true", a relocatable partial export will include email notifiers.

Positional arguments

fileName

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->export(<fileName>, {<optionals>});`

Examples

```
$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

```
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`.

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 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.
	
	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 object IDs that will be returned. If omitted, default behavior returns IDs for the first 1000 objects matching the query. If "0" is specified, the ID of every matching object is returned.

Arguments	Descriptions
numObjects	<full object count> 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 "full objects" 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.
objectType	This argument specifies the type of object to find. **Values include:** artifact, artifactVersion, credential, directoryProvider, emailconfig, emailNotifier, formalParameter, job, jobStep, logEntry, plugin, procedure, procedureStep, project, property, repository, resource, resourcePool, schedule, state, stateDefinition, transition, transitionDefinition, workflow, workflowDefinition, workspace
select	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.
sort	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.

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.

```
my @filterList;
push (@filterList, {"propertyName" => "jobName",
                        "operator" => "like",
                        "operand1" => "%-branch-%"});
push (@filterList, {"propertyName" => "jobName",
                        "operator" => "like",
                        "operand1" => "branch-%"});
my $result = $cmdr->findObjects('job',
        {filter => [
     { operator => 'or',
         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.

```
# 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
                   {
                       propertyName => "modifyTime",
                           operator => "lessOrEqual", # Give me all dates before
                           operand1 => "2011-11-010T00:00:00.000Z" # Arbitrary date
                   },
                   #filter 2
                   {
                       operator => '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
              {
                  propertyName => "modifyTime",
                      operator => "lessOrEqual", # Give me all dates before
                      operand1 => "2011-11-010T00:00:00.000Z" # Arbitrary date
              },
              #filter 2
              {
                  operator => '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'.

```
$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'.)

```
$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.

```
$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`.

Arguments	Descriptions
`objectId`	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.
`select`	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.

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`:

```
# 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 => "2010-01-20T00:00:00.000Z"},
                   {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 2

Code example using a Boolean filter:

```
my $xpath = $N->findObjects('project', {
        filter => {operator => 'and',
               filter => [{propertyName => 'projectName',
                           operator => 'contains',
                           operand1 => $projectBase},
                          {propertyName => 'description',
                           operator => 'equals',
                           operand1 => 'foo'}]}});
```

ectool

Not supported.

Back to Top

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.

Arguments	Descriptions
`batchSize`	<*batch size*> 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. **Note:** The `batchSize` argument is applicable to full import operations only.
`disableSchedules`	<*Boolean flag* - `0|1|true|false`> If set to 1, imported schedules will be disabled. This argument can modify imported schedules after import and before they are used to start a job.

Arguments	Descriptions
file	<*localFileName*> This is the path to a file on the client to import. The file is uploaded from the client to the server. The specified <file> 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 file 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.
fileName	<*remoteFileName*> 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.
force	<*Boolean flag* - 0|1|true|false> This argument can be used to replace a single object if it already exists at the specified property path.
path	<*property path*> Use this argument to import a single object to a new location. For example, if a procedure was exported from "project A", this argument allows you to import it into "project B", but only if the export used the path option also.
preserveId	If performing a *partial* import, using this option preserves the original job ID or workflow ID.

Positional arguments

fileName

Response

None or a status OK message.

ec-perl

syntax examples: `$cmdr->import(<fileName>, {...});`

`$cmdr->import({file => <localFileName>, ...);`

Examples

```
$cmdr->import("/opt/TestProg.xml");

$cmdr->import({file => "c:\r.xml", path => "/projects[Test]");
```

ectool

syntax examples: `ectool import <remoteFileName> ...`

`ectool import --file <localFileName>`

Examples

```
ectool import /mnt/backups/fullBackkup.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:

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:

artifactVersion

A "concrete" version of an artifact that contains a collection of files stored in the artifact repository.

Contents:

dependentArtifacts

description

groupId

lastModifiedBy

qualifier

repositoryName

retrievers

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

createTime

description

directoryProviderId

domainName

emailAttribute

enableGroups

fullUserNameAttribute

groupBase

groupMemberAttributes

groupMemberFilter

groupNameAttribute

groupSearchFilter

lastModifiedBy

managerDn

modifyTime

name

owner

position

propertySheetId

providerIndex

providerName

providerType

realm

url

useSSL

userBase

userNameAttribute

userSearchFilter

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

groupList

group*

findUserTest

testResult

details

userList

userInfo*

email

fullUserName

mutable

providerName

userAuthenticationTest

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

configName

container

createTime

description

destinations

emailNotifierId

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:

createTime

description

gatewayDisabled

gatewayId

gatewayName

hostName1

hostName2

lastModifiedBy

modifyTime

owner

port1

port2

propertySheetId

resourceName1

resourceName2

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:

abortedBy
abortStatus
actualParameters*
callingState
combinedStatus
createTime
credentialName
deleted
directoryName
elapsedTime
errorCode
errorMessage
external
finish
jobId
jobName
jobStep*
lastModifiedBy
launchedByUser
licenseWaitTime
liveProcedure
liveSchedule
modifyTime
outcome
owner
priority
procedureName
projectName
propertySheet
propertySheetId
resourceWaitTime
runAsUser
scheduleName
start
status
steps
totalWaitTime
workspaceWaitTime

jobStep

Contains information to define or locate a job step. Notice that the `calledProcedure` element (subcontainer element) can contain multiple `jobStep` elements.

Contents:

abortedBy
abortStatus
actualParameters
alwaysRun
assignedResourceName
broadcast
calledProcedure
    jobStep*
outcome
owner
parallel
postExitCode
postLogFileName
postProcessor
precondition
procedureName

combinedStatus
command
condition
createTime
delayUntil
elapsedTime
errorCode
errorHandling
errorMessage
exclusive
exclusiveMode
exitCode
external
finish
hostName
jobId
jobName
jobStepId
lastModifiedBy
licenseWaitTime
liveProcedure
liveProcedureStep
logFileName
modifyTime
projectName
propertySheetId
releaseExclusive
releaseMode
resourceName
resourceWaitTime
retries
runAsUser
runnable
runTime
shell
start
status
stepName
subprocedure
subproject
timeLimit
timeout
totalWaitTime
waitTime
workingDirectory
workspaceName
workspaceWaitTime

license

Contains information to specify the Commander license.

Contents:

createTime
customerName
evaluation
expirationDate
featureName

gracePeriod

lastModifiedBy

licenseId

modifyTime

owner

productName

property*

propertySheet*

signature

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

resourceName

shell

stepCount

stepLimit

trusted

useSSL

workspaceName

zoneName

resourcePool

Contains elements to define a resource pool.

Contents:

autoDelete

createTime

description

lastModifiedBy

lastResourceUsed

modifyTime

orderingFilter

owner

propertySheetId

resourceNames

resourcePoolDisabled

resosurcePoolId

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:

createTime

description

formalParameters

index

lastModifiedBy

modifyTime

owner

projectName

propertySheetId

startable

stateDefinitionId

stateDefinitionName

subprocedure

subproject

substartingState

subworkflowDefinition

workflowDefinitionName

step

Contains elements to specify or define a step.

Contents:

actualParameters

alwaysRun

attachedCredentials

attachedParameters

broadcast

command

condition

createTime

credentialName*

description

errorHandling

exclusive

exclusiveMode

postLogFileName

postProcessor

precondition

procedureName

projectName

propertySheetId

releaseExclusive

releaseMode

resourceName

shell

stepId

stepName

subprocedure

lastModifiedBy
logFileName
modifyTime
owner
parallel
subproject
timeLimit
timeLimitUnits
workingDirectory
workspaceName

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

lastModifiedBy

modifyTime

owner

projectName

propertySheetId

stateDefinitionName

targetState

transitionDefinitionId

transitionDefinitionName

trigger

workflowDefinitionName

user

Contains information about the current user.

Contents:

createTime

email

fullUserName

groups

lastModifiedBy

modifyTime

mutable

owner

propertySheetId

providerName

userId

userName

workflow

Contains elements about a running or completed workflow.

Contents:

activeState

callingState

completed

createTime

deleted

elapsedTime

finish

lastModifiedBy

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.

Returned element	Type	Description/Value
abortStatus	enum	Possible values are: abort|force_abort
abortedBy	string	The name of the user who aborted the job.

Returned element	Type	Description/Value
aclEntryId	number	The unique Commander-generated ID for this aclEntry object.
active	boolean	<*Boolean flag* - 0|1|true|false>—If set to "true", the state of the workflow is active.
activeCalls	subcontainer	A container element within the serverStatus element. activeCall describes an API currently running on the server.
activeState	string	The name of the activeState on the workflow object.
actualParameters	propertySheet	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. For the workflow feature, these are the parameters that were passed when the workflow was started.
actualParameterId	number	The unique Commander-generated ID for this actual parameter object.
actualParameterName	string	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.
admin	boolean	<*Boolean flag* - 0|1|true|false>—If set to "true", the this is an "admin" license.
adminLicenseLastUse	date	The time at which the admin license was last used.
adminLicenseUser	string	The name of the user who is currently licensed as the "admin" user.
agentDrivePath	string	Drive-letter-based path used by Windows agents to access the workspace in steps.

Returned element	Type	Description/Value
agentUncPath	string	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.
agentUnixPath	string	UNIX path used by UNIX agents and Linux Commander Web servers to access the workspace.
agentState	subcontainer	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|alive|down
alive	boolean	Refers to the agent state or status.
alwaysRun	boolean	<*Boolean flag* - 0|1|true|false> - If set to 1, indicates this step will run even if the job is aborted before the step completes. Defaults to "false".
api	string	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.
apiMonitor		A server object that tracks API active and recent calls, as well as the total number of calls since server startup.
artifactCacheDirectory	string	The directory on the agent host where retrieved artifacts are stored.
artifactId	number	The unique Commander-generated ID for this artifact object.
artifactKey	string	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.
artifactName	string	The name of the artifact.
artifactsDirectory	string	The directory in the workspace where you can put files to view, using a report link.
artifactVersionId	string	The unique Commander-generated ID for this artifact version object.

Returned element	Type	Description/Value
`artifactVersionName`	name	The name of the artifact version. 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.
`artifactVersionNameTemplate`	string	A template for the names of artifact versions published to this artifact. Over-rides the ***global***`artifactVersionNameTemplate`. The global setting can be manipulated in the Server Settings page (Administration > Server, select the Settings link).
`artifactVersionState`	enum	Possible values are: `available|publishing|unavailable`
`assignedResourceName`	string	The name of the resource assigned to the step by the step scheduler.
`attachedCredentials`	list	The names of the credentials attached to the specified object.
`attachedParameters`	string	These are credential parameters that were attached to a step.
`author`	string	The author of the plugin.
`autoDelete`	boolean	<*Boolean flag* - `0|1|true|false`> - If "true", the resource pool is deleted when the last resource is removed or deleted.
`batchRequests`	string	A setting in the database configuration that determines whether or not to batch SQL queries when making a request to the database.
`batchSize`	string	The number of objects imported before being committed to the database.
`beginDate`	string	<*yyyy-mm-dd*> The date the schedule is set to begin.

Returned element	Type	Description/Value
broadcast	boolean	<*Boolean flag* - 0|1|true|false> - Used for command steps, this flag is used to run the same step on several resources at the same time. The step is “broadcast” to all resources listed in the resourceName argument. Defaults to "false".
buildNumber	string	User-defined build number component of the version attribute for the artifact version.
call	subcontainer	A subcontainer returned on activeCall and recentCall elements returned by the serverStatus API. call contains information specific to each API call on the server.
callId	number	A unique Commander-generated identifier for this particular call.
callingState	string	The full property path to the "calling state", which can appear on subjobs and subworkflows of a workflow.
calledProcedure	list	A subcontainer element within the jobStep element. The calledProcedure element can contain multiple jobStep elements.
category		(currently not used)
changePermissionsPrivilege	enum	Possible values are: allow|deny|inherit
code	enum	Script to execute the functions for a step—passed to the step's shell for execution.
combinedStatus	enum	More inclusive step status output - this value may combine up to three sub-elements: status|message|properties
command	string	The command to run steps - for command steps.
commonGroupNameAttribute	string	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.
completed	boolean	<*Boolean flag* - 0|1|true|false> - If "true", the workflow is completed and no additional transactions will be evaluated.

Returned element	Type	Description/Value
completeUserName	string	A SQL server-specific tag that includes the user's name and the user's domain name.
concurrentResources	object	A subcontainer element that includes information about "in use" and "maximum licensed" hosts and proxied hosts for the licenseUsage API command.
concurrentSteps	number	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.)
concurrentUsers	object	A subcontainer element that includes information about the admin license, "in use" licenses, and the maximum number of licenses for the licenseUsage API command.
condition	string	**For steps:** If empty or non-zero, the step will run. If set to "0", 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. **For email notifiers:** Mail sent if the condition evaluates to "true". The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is "false" or "0". If no condition is specified, the notification is ALWAYS sent.
configName	string	The name of the configuration.
container	string	An object ID for a "container" that contains formal parameters. In another context, this is typically the type and name of the workflow or job with a corresponding ID.
containerName	string	The name of the container.
createTime	date	The time when this object was created.
credentialId	number	The unique Commander-generated ID for this credential object.

Returned element	Type	Description/Value
credentialName	string	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.
customDatabaseDialect	string	Class name for the Hibernate dialect. The server chooses an appropriate dialect based on databaseType or this can be part of the custom specification.
customDatabaseDriver	string	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.
customDatabaseUrl	string	The JDBC URL to use. The server will compose an appropriate URL or this can be part of the custom specification.
customerName	string	The name of a company and/or group name with a company that is using ElectricCommander.
databaseDialect	string	Class name for the Hibernate dialect (the server chooses an appropriate dialect based on databaseType).
databaseDriver	string	Class name of the JDBC driver (the server will choose an appropriate driver based on databaseType).
databaseName	string	The name of the database the Commander server is using.
databaseType	enum	Possible values are: builtin|mysql|oracle|postgresql|sqlserver
databaseUrl	string	The JDBC URL to use (the server will compose an appropriate URL).
defaultValue	string	This value is used for the formal parameter if a value is not supplied by the caller.

Returned element	Type	Description/Value
delayUntil	date	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.
deleted	byte	The object was marked for background deletion. Possible values are "0" or "1". Default is "0" (not set).
dependentArtifacts	string	A space-separated list of artifacts.
description	string	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> <dl> <font> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr>
destinations	string	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.
details	string	A string containing details about agent status.
directoryName	string	The name of the job's directory within each workspace for a job.
directoryProviderId	number	The unique Commander-generated ID for this directory provider object.
domainName	string	The name of the domain from which the Active Directory server(s) are automatically discovered.
elapsedTime	number	The number of milliseconds between the start and end times for the job or job step - or a workflow.
email	string	The user's email address.
emailAttribute	string	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.
emailConfigId	number	The unique Commander-generated ID for this email configuration object.

Returned element	Type	Description/Value
emailConfigName	string	The name of the email configuration.
emailNotifierId	number	The unique Commander-generated ID for this email notifier object.
enableGroups	boolean	Determines whether or not to enable external groups for the directory provider.
endDate	string	<*yyyy-mm-dd*> The date this schedule is set to end.
errorCode	enum	Displays the error code, identifying which error occurred.
errorHandling	enum	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.
errorMessage	string	A description of the error.
evaluation	boolean	Determines whether or not this license is an evaluation copy only.
eventType	enum	Possible values are: onCompletion|onStart "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".

Returned element	Type	Description/Value
`exclusive`	boolean	<*Boolean flag* - `0|1|true|false`> - If set to 1, indicates this step should acquire and retain this resource exclusively. Defaults to "false".
`exclusiveJobId`	number	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.
`exclusiveJobName`	string	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.
`exclusiveJobStepId`	number	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.
`exclusiveJobStepName`	name	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.
`exclusiveMode`	enum	Possible values are: `none|job|step|call` See `exclusive`
`executePrivilege`	enum	Possible values are: `allow|deny|inherit`
`exitCode`	number	The step's exit code.
`expandable`	boolean	<*Boolean flag* - `0|1|true|false`> Determines whether the property value will undergo property expansion when it is fetched. Default is "true".
`expansionDeferred`	boolean	<*Boolean flag* -`0|1|true|false`> Default is "false," which means the formal parameter is expanded immediately.
`expiration`	date	The date when a user license expires.
`expirationDate`	date	The date when a license expires.
`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.

Returned element	Type	Description/Value
featureName	string	The name of the licensed feature. Possible features include: Server
findGroupsTest	subcontainer	For the testDirectoryProvider API, this element provides information on which groups the user is a member.
findUserTest	subcontainer	For the testDirectoryProvider API, this element contains specific information about the user.
finish	date	The time the job or workflow completed.
formalParameterId	number	The formal parameter's ID.
formalParameterName	string	The name of the procedure's parameter, containing a credential reference.
formalParameters	string	The parameters that must be supplied when entering the state (similar to formal parameters on a procedure).
formattingTemplate	string	Specifies a template for formatting email messages when an event [notification] is triggered by the emailNotifier.
fullUserName	string	The user's full name - not his or her nickname.
fullUserNameAttribute	string	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.
gatewayDisabled	boolean	<*Boolean flag* -0|1|true|false> If "true", the gateway is disabled.
gatewayId	number	The Commander-generated ID number for this gateway.
gatewayName	string	The name of the gateway.
gateways	list	A space-separated list of gateway names.
gracePeriod	number	The number of days available after the Commander license expires.
groupBase	string	This string is prepended to the basedn to construct the directory DN that contains group records.

Returned element	Type	Description/Value
groupId	number	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.
groupList	list	For the testDirectoryProvider API, this element contains zero or more groups returned after querying existing groups known to the directory provider.
groupMemberAttributes	string	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.
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 groupOfNames or uniqueGroupOfNames records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: "{0}" is replaced with the full user record DN, and "{1}" is replaced with the user's account name.
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.

Returned element	Type	Description/Value
hostName1	string	For gateways: The name Resource 2 uses to communicate with Resource 1. If "blank", the *Agent Host Name* attribute in Resource 1's definition is used at runtime.
hostName2	string	For gateways: The name Resource 1 uses to communicate with Resource 2. If "blank", the *Agent Host Name* attribute in Resource 2's definition is used at runtime.
hostOS	string	The full name of the host operating system, plus its version. However, if this host is a proxy, the value is "proxied".
hostPlatform	string	Examples for "platform" are: Windows, Linux, HPUX, and so on. However, if this host is a proxy, the value is "proxied".
index	number	The numeric index of the transition that indicates its order in the list of transitions in a state definition.
interval	string	The repeat interval for starting new jobs.
intervalUnits	enum	Possible values are: `hours|minutes|seconds|continuous` If set to `continuous`, Commander creates a new job as soon as the previous job completes.
inUseHosts	number	The number of hosts (agents) currently in use.
inUseLicenses	number	The number of user licenses currently in use.
inUseProxiedHosts	number	The number of proxy target hosts currently in use.
jobId	number	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.
jobName	string	The name of the job.
jobNameTemplate	string	Template used to determine the default name of jobs launched from a procedure.
jobStepId	number	The unique identifier for a job step, assigned automatically when the job step is created.
jobStepName	string	The name of the job step.

Returned element	Type	Description/Value
label	string	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 "label" tag.
lastMessage	string	Element returned by the serverStatus API showing the last message the server received.
lastModifiedBy	string	Shows who (generally a user name) last modified the object.
lastResourceUsed	string	The name of the most recently used resource from the pool.
lastRunTime	date	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.
lastUse		Returned element in the concurrentUsers subcontainer (for the licenseUsage API), providing the last time a specific user accessed Commander.
launchedByUser	string	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.
licenseId	number	The unique Commander-generated ID for this license.
licenseWaitTime		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.
liveProcedure	string	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.
liveProcedureStep	string	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.

Returned element	Type	Description/Value
liveSchedule	string	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.
liveWorkflowDefinition	string	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.
local	boolean	<*Boolean flag* -0|1|true|false> If "true", this object is local.
logEntryId	number	The Commander-generated ID number for the log entry record.
logFileName	string	A custom log file name produced by running the step. By default, ElectricCommander assigns a unique name for this file.
longestCall	string	Provides the API call that took the longest time.
mailFrom	string	The email address used as the email sender address for notifications.
mailHost	string	The name of the email server host.
mailPort	number	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.
mailProtocol	string	This is either SSMTP or SMTP (not case-sensitive). The default is SMTP.
mailUser	string	This can be an individual or a generic name like "Commander" - name of the email user on whose behalf Commander sends email notifications.
majorMinorPatch	string	major.minor.patch component of the version attribute for the artifact.

Returned element	Type	Description/Value
managerDn	string	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. **Note**: This user does not need to be an admin user with modify privileges.
maxConcurrentSteps	number	The maximum number of steps that can run at the same time per the provisions of your Commander license.
maxHosts	number	The maximum number of hosts licensed for resource use.
maxLicenses	number	The maximum number of licenses available for users.
maxProxiedHosts	number	The maximum number of available licenses for proxy hosts.
message	string	A user-readable diagnostic message associated with an error.
messages	list	Multiple error or diagnostic messages.
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"`.
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.

Returned element	Type	Description/Value
mutable	boolean	If "true," the member list of this group is editable within Commander via the web UI or the modifyGroup API.
name	string	The name of the directory provider.
notifierName	string	The name of the email notifier.
objectId	number	An object identifier returned by findObjects and getObjects. This value is a "handle" only for passing to API commands. The internal structure of this value is subject to change - do not parse this value.
objectName	string	The name of the object.
objectType	enum	The type of object being described, for example: project, procedure, step, and so on.
orderingFilter	string	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.
outcome	enum	Possible values for outcome: **Note:** 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.
owner	string	The person (user name) who created the object.
parallel	boolean	<*Boolean flag* - 0|1|true|false> - If set, indicates this step should run at the same time as adjacent steps marked to run as parallel also. Defaults to "false".
password	string	The password matching the specified user name.
path	string	The property path that specifies the object to use.

Returned element	Type	Description/Value
pingToken	number	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.
pluginId	number	The unique Commander-generated ID for the plugin object.
pluginKey	string	The name of the plugin as displayed on the Commander Plugin Manager web page.
pluginName	string	The name of the plugin - the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.
pluginVersion	string	The version of the plugin being described.
pools	list	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.
port	number	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.
port1	number	The port number used by *Gateway Resource1* - default is to the port number used by the resource.
port2	number	The port number used by *Gateway Resource2* - default is to the port number used by the resource.
position	number	Used to reorder a Commander object. For example, if reordering directory providers: the provider name is moved to a position just before this provider. "Blank" means move the provider to the end of the provider list.
postExitCode	number	The step's post processor exit code.
postLogFileName	string	The log file name produced by this step’s post processor.

Returned element	Type	Description/Value
postProcessor	string	This program looks at the step output to find errors and warnings. Commander includes a customizable program called “postp” 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).
precondition	string	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.
principal	string	The user or project principal from the session that was active when the event occurred.
principalName	string	This is either a user or a group name.
principalType	enum	Possible values are: group|user
priority	enum	Possible values are: 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.
procedureId	number	The unique Commander-generated procedure ID.
procedureName	string	The name of the procedure - may be a path to the procedure.
productName	string	The name of the product with the licensed feature. Possible products include: ElectricCommander

Returned element	Type	Description/Value
project	name	The name of the project associated with the plugin.
projectId	number	The unique Commander-generated project ID.
projectName	string	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.
promoted	boolean	<*Boolean flag* - 0|1|true|false> The new value of the promoted flag for the specified plugin. Default is "true", which means the plugin will be promoted. If you want to demote the plugin, use the value of “0” or false.
propertyId	number	The unique Commander-generated property ID.
propertyName	string	The name of the property. It may be a relative or absolute property path, including "my" paths such as "/myProject/prop1".
propertySheetId	number	The unique identifier for a property sheet, assigned automatically when the property sheet is created.
protocolVersion	sting	The server API protocol version. For example, the server accepts messages from ectool and ec-perl.
providerIndex	number	The index that specifies the search order across multiple directory providers. For example: 2 LDAP providers, one with index "0" and one with index "1" means the providers will be searched in that numerical order.
providerName	string	The LDAP or Active Directory provider name.
providerType	enum	Possible values are: ldap|activedirectory
proxyCustomization	string	Perl code customizing how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources.
proxyHostName	string	The name or IP address of the computer containing the ElectricCommander Agent used for a proxy resource.

Returned element	Type	Description/Value
proxyPort	number	The ElectricCommander agent port number for a proxy resource.
proxyProtocol	string	Protocol for communicating with the proxy target. Defaults to ssh. (This argument is not exposed in the ElectricCommander Web Interface at this time.)
publisherJobId	number	The Commander-generated ID for the job that published the artifact version.
publisherJobName	name	The name of the job that published the artifact version.
publisherJobStepId	number	The Commander-generated ID for the job step that published the artifact version.
qualifier	string	User-defined qualifier component of the version attribute for the artifact.
readPrivilege	enum	Possible values are: allow|deny|inherit
realm	string	The realm of the LDAP directory provider—used to create unique user names when there are multiple providers.
recentCall	subcontainer	A subcontainer element on the serverStatus API - a call no longer active (completed). The API monitor keeps track of the 10 most recent calls.
releaseExclusive	boolean	<*Boolean flag* - 0|1|true|false> Declares whether or not this step will release its resource, which is currently held exclusively.
releaseMode	string	Possible values are: none|release|releaseToJob
remoteAddress	string	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.
repositoryDisabled	boolean	<*Boolean flag* - 0|1|true|false> Determines whether the repository is disabled. Default is "false".
repositoryId	number	The Commander-generated ID for the artifact repository.

Returned element	Type	Description/Value
repositoryIndex	integer	The order of the repository within a list of repositories.
repositoryName	string	The name of the artifact repository.
repositoryNames	list	A list of one or more repository server names—each repository name listed on a "new line".
required	boolean	<*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.
resourceDisabled	boolean	<*Boolean flag* - 0|1|true|false> If set to 1, Commander will not start new steps on this resource. Defaults to "false".
resourceId	number	The unique Commander-generated ID for this resource.
resourceName1	string	The name for the first of two resources required to create a gateway. "Spaces" are NOT allowed in a resource name.
resourceName2	string	The name for the second of two resources required to create a gateway. "Spaces" are NOT allowed in a resource name.
resourceName	string	The name of a resource.
resourceNames	string	A list of strings that refer to resources that belong to the pool. Names that do not refer to existing resources are ignored.
resourcePoolDisabled	boolean	<*Boolean flag* - 0|1|true|false> If set to 1, Commander will not use resources in this pool. Defaults to "false".
resourcePoolId	number	The unique ID number for a resource pool.
resourcePoolName	name	The name of the resource pool.
resources	string	A space-separated list of resource names.
resourceUsageId	number	The unique ID number of the resource usage record.

Returned element	Type	Description/Value
resourceWaitTime		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.
retries	number	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 the workspace for the step was unavailable.
retrievers	list	A collection of retrieve elements that can contain a jobName, jobId, and/or a jobStepId element(s).
runAsUser	string	The name of the user being impersonated in this job.
runnable	date	The time when the step became runnable.
runningSteps		The number of steps running at the same time.
runtime	number	The number of milliseconds the step command spent running on a resource.
scheduleDisabled	boolean	<*Boolean flag* - 0|1|true|false> If set to 1, ElectricCommander does not start any new jobs from the schedule. Defaults to "false".
scheduleId	number	The unique Commander-generated ID for the schedule.
scheduleName	string	The name of the schedule - may be a path to the schedule.
schemaVersion	number	The Commander server's database schema version.
serverState	enum	Possible values are: bootstrap, databaseConfiguration, databaseConnection, databaseSchema, running, failed, stopping, importFailed
severity	enum	Possible values are: INFO|WARN|ERROR

Returned element	Type	Description/Value
shell	string	Where shell is the name of a program used to execute commands contained in the "command" field. Normally, this file is a command shell, but it could be any other command line program. The default is "cmd /q /c" for a Windows agent and "sh -e" for a UNIX agent. This is applicable to command steps only.
signature	string	The digital signature on this license.
start	date	The time this job or workflow began executing.
startable	boolean	"True" means this state definition can be the initial state of an instantiated workflow.
startingState	string	The initial state of the workflow.
startTime	string	Formatted hh:mm, using the 24-hour clock. Using this schedule, Commander starts creating jobs at this time on the specified days.
stateDefinitionId	number	The unique Commander-generated ID for this state definition object.
stateDefinitionName	string	The name of the state definition.
stateId	number	The unique Commander-generated ID for this state object.
statementCacheSize	string	The number of MS SQL statements cached in the database.
stateName	string	The name of the state.
status	enum	Possible values for status: pending - The job is not yet runnable—it is waiting for other steps to complete first. 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.
stepCount	number	The number of executing steps on this resource.
stepErrorCode	enum	Agent error messages.
stepId	number	The unique Commander-generated ID for the step.

Returned element	Type	Description/Value
stepLimit	number	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.)
stepName	string	The name of the step - may be a path to the step.
steps		The list or number of steps in a job.
stopTime	string	Formatted hh:mm, using the 24-hour clock. ElectricCommander 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.
subject	string	Refers to the object the event concerns (similar to container).
subjectName	string	The name of the subject/object.
subjob	string	The name of the subjob.
subprocedure	string	The name of the nested procedure called when a step runs. If a subprocedure is specified, command or commandFile options are not necessary.
subproject	string	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.
substartingState	sting	Name of the starting state for the workflow launched when the state is entered.
subworkflow	string	The name of the subworkflow.
subworkflowDefinition	string	The name of the subworkflow definition.
targetState	string	The target state for the transition definition.
testResult	enum	Possible values are: success|skipped|failure
time	date	The time of day to invoke this schedule's procedure (24-hour clock, for example, 17:00). For a logEntry response, time indicates the time at which data was written to the log.

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.

Returned element	Type	Description/Value
type	string	The "type" is 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.
url	string	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 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). **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 https://host:8200/.
userAuthenticationTest	subcontainer	For the testDirectoryProvider API, this element authenticates the user.
userBase	string	The string prepended to the basedn to construct the directory DN that contains user records.
userId	number	The unique Commander-generated ID for the user.
userInfo		findUserTest container element includes a userList subcontainer that may include multiple userInfo tags, each of which describes a user (including full name, email address, and provider name).
userList	list	findUserTest container element includes a userList subcontainer that may include one or more userInfo tags.
userName	string	The full name of the user. For Active Directory and LDAP, the name may be user@domain.
userNameAttribute	string	The attribute in a user record that contains the user's account name.

Returned element	Type	Description/Value
`userSearchFilter`	string	The LDAP query 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	<*Boolean flag* - `0|1|true|false`> If true, the subtree below the user base was recursively searched.
`userSettingsId`	number	The unique Commander-generated ID for the user settings.
`useSSL`	boolean	<*Boolean flag* - `0|1|true|false`> This flag is used to specify using SSL to communicate with your Active Directory servers.
`value`	string	For a string property, this is the value of the property. For a sheet property, this argument is invalid.
`version`	string	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`
`waitReason`	string	Possible values are: `license`, `resource`, or `workspace` Generally, this objects are unavailable, causing a longer wait time for availability.
`waitTime`	number	The number of milliseconds the step spent between runnable and running (for example, waiting for a resource).
`weekDays`	string	Restricts the schedule to specified days of the week. Days of the week are separated by spaces. English names "Monday", "Tuesday", and so on.
`workflowDefinitionId`	number	The unique Commander-generated ID for this workflow definition.
`workflowDefinitionName`	string	The name of the workflow definition.

Returned element	Type	Description/Value
workflowId	number	The unique Commander-generated ID for this workflow object.
workflowName	string	The name of this workflow.
workflowNameTemplate	string	Template used to determine the default names for workflows launched from a workflow definition.
workingDirectory	string	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. **Note:** If running a step on a proxy resource, this directory must exist on the proxy target.
workspaceDisabled	boolean	<*Boolean flag* - 0|1|true|false> - If "true," the workspace is disabled.
workspaceId	number	The unique Commander-generated ID for the workspace.
workspaceName	string	The name of the workspace.
workspaceWaitTime		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.
zoneId	number	The Commander-generated ID for this zone.
zoneName	string	The name of the zone.

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.

Term	Description
access control	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.
ACE (Access Control Entry)	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.
ACL (Access Control List)	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.
actual parameter	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 "formal parameters": formal parameters define parameters a procedure is expecting, and actual parameters provide values to use at run-time.
admin	"admin" 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.
agent	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.
artifact	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.
artifact key	An artifact key is an identifier for an artifact and the "key" component of the artifact name.

Term	Description
artifact repository	See repository.
artifact version	An artifact version is a collection of 0 to N files that were published to an artifact repository.
backingstore	The backingstore is the directory on the repository server where artifact versions are stored. By default, the backingstore is the <datadir>/repository-data directory in the repository installation—this default setting can be changed.
compression	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.
continuous integration	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.
credential	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*.
custom property	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.
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>   <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`

Term	Description
diagnostic extract	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 "2770" is the unique identifier for the step. Other postprocessors you may use can have a different filename configuration.
dynamic credential	Dynamic credentials are captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded.
ec-perl	*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.
ectool	*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.
ElectricAccelerator	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.
ElectricSentry	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.
email configuration	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.
email notifier	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.
Event log	See log (s).
Everyone	A special intrinsic access control group that includes **all** users.

Term	Description
filter	Two filter categories: • Intrinsic filters - these filters provide a convenient way to access certain well-defined fields for jobs. • 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).
formal parameter	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 "actual parameters": formal parameters define the kinds of parameters a procedure is expecting, and actual parameters provide values to use at run-time.
gateway	To communicate with a resource, workspace, or artifact repository server in another zone, a "gateway" 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).
group	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.
impersonation	Impersonation is a mechanism that allows a job step to execute under a particular login account (the Commander agent "impersonates" a particular user during the execution of that step). Impersonation is implemented using credentials.
inheritance	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.
intrinsic property	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`.
job	A job is the output associated with invoking a Commander procedure. A new job is created each time you **run** (execute) a procedure.

Term	Description
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.

Term	Description
local group	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.
local user	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.
log(s)	ElectricCommander provides a log for events generated anywhere in the system, including jobs and workflows. **Note:** 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. • 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. • 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. • To see *only* events for a specific object, select the Search tab to go to the Define Search page. For example: You can select the Object Type, "Log Entry", then click the **Add Intrinsic Filter** link. Select the down-arrow where you see "Container" auto-populated and select "Container Type. Use the "equals" operator, then select the next down-arrow to choose an object. Click **OK** to start the search. See the Event Log topic for more information.
matcher	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.
misfire policy	A misfire policy allows you to manage how a schedule resumes in cases where the normal scheduled time is interrupted. Available options are: `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). See schedule.

Term	Description
parameter	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.
plugin	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.
polling frequency	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.
pool	Also known as "resource pool". 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.
postp	*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 "error." *postp* also supports several useful command-line options, and it can be extended using "matchers" to handle environment-specific errors. See matcher.
postprocessor	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 "extend" postp. See matcher.
preflight build	A preflight build provides a way to build and test a developer's changes before those changes are committed. A "post-commit" 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.

Term	Description
privileges	Commander supports four privilege types (for access control/security) for each object: • Read - Allows object contents to be viewed. • Modify - Allows object contents (but not its permissions) to be changed. • 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. • Change Permissions - Allows object permissions to be modified.
procedure	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 "subprocedure" (also known as a "nested" procedure). The step can pass arguments to the subprocedure.
project	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: • 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. • 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.
project principal	*Project principal* is a special user ID associated with each project. If a project name is "xyz," the project principal for that project is `"project: xyz"` (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.

Term	Description
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.
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 "property sheet" 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.

Term	Description
quiet time	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 "back-out" 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.

reports	ElectricCommander provides multiple reports and custom report capabilities to help you manage your build environment.
	• Real-time reports - filtered view of your workload in real-time
	• Build reports - summary reports produced at the end of a build and attached to the job
	• Batch reports - summaries of your build environment with trends over time, two types:
	  • Default Batch reports - automatically installed during ElectricCommander installation and scheduled to run daily (Cross Project Summary, Variant Trend, Daily Summary, Resource Summary, Resource Detail)
	  • 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)
	• Custom reports - your choice to create and add at any time

repository or repository server	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 `<datadir>/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.

Term	Description
resource	A resource specifies an agent machine where job steps can be executed. Resources can be grouped into a "pool", also know 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 definition.

Term	Description
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

stored credential	*Stored credentials* 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 "nesting" 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 `"echo $[status]"`, and when the step executes there is a property named `"status"` with value `"success"`, the actual command executed will be `"echo success"`.
system object	This is a special object whose access control lists are used to control access to some ElectricCommander internals. System objects are: `admin`, `artifactVersions`, `directory`, `emailConfigs`, `forceAbort`, `licensing`, `log`, `plugins`, `priority`, `projects`, `repositories`, `resources`, `server`, `session`, and `workspaces`.

Term	Description
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.

Term	Description
workflow	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 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.

workflow name template	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`

workspace	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.
workspace root	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.
zone	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).