Electric Cloud

ElectricFlow API Guide

Electric Cloud, Inc.

35 South Market Street, Suite 100
San Jose, CA 95113
www.electric-cloud.com

ElectricFlow version 7.0.3

Copyright © 2002–2017 Electric Cloud, Inc. All rights reserved.

Published 6/30/2017

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, ElectricFlow, and ElectricInsight are registered trademarks or trademarks of Electric Cloud, Incorporated.

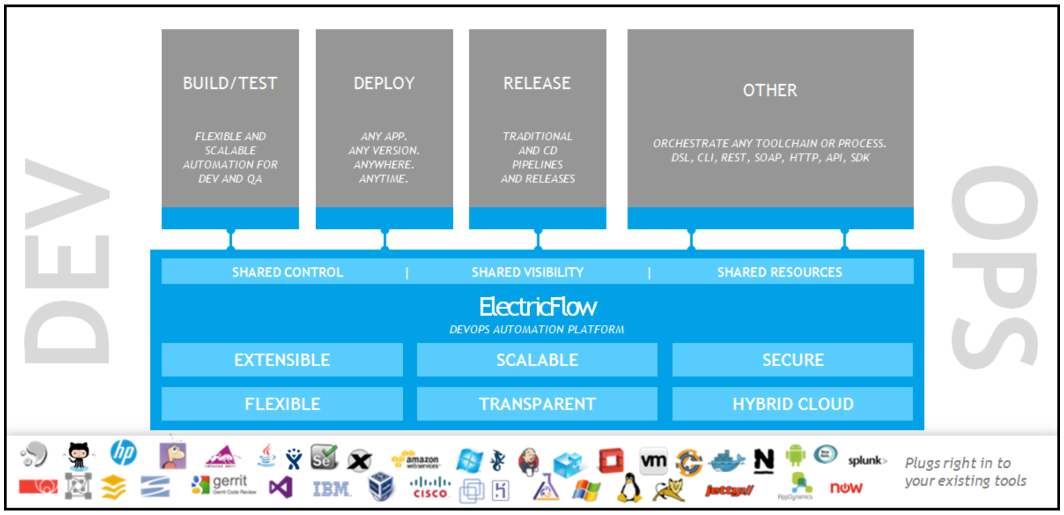
Electric Cloud products—ElectricAccelerator, ElectricCommander, ElectricInsight, ElectricFlow Deploy, ElectricFlow Release, and Electric Make—are commonly referred to by their “short names”—Accelerator, Commander, Deploy, eMake, Flow, Insight, and Release—throughout various types of Electric Cloud product-specific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

Other product names mentioned in this guide may be trademarks or registered trademarks of their respective owners and are hereby acknowledged.

Contents

Introduction to ElectricFlow

ElectricFlow™ is an enterprise-grade DevOps Release Automation platform that simplifies provisioning, build and release of multi-tiered applications. Our model-driven approach to managing environments and applications allows teams to coordinate multiple pipelines and releases across hybrid infrastructure in an efficient, predictable and auditable way.


DEV
BUILD/TEST
FLEXIBLE AND SCALABLE AUTOMATION FOR DEV AND QA
DEPLOY
ANY APP. ANY VERSION. ANYWHERE. ANYTIME.
RELEASE
TRADITIONAL AND CD PIPELINES AND RELEASES
OTHER
ORCHESTRATE ANY TOOLCHAIN OR PROCESS. DSL, CLI, REST, SOAP, HTTP, API, SDK
OPS
SHARED CONTROL | SHARED VISIBILITY | SHARED RESOURCES
ElectricFlow
DEVOPS AUTOMATION PLATFORM
EXTENSIBLE
SCALABLE
SECURE
FLEXIBLE
TRANSPARENT
HYBRID CLOUD
Plugs right in to your existing tools

Web-Based System

At its core, ElectricFlow automation platform is a web-based system for automating and managing the build, test, deployment, and release process. It provides a scalable solution, solving some of the biggest challenges of managing these "back end" software development tasks, including these challenges:

- Time wasted on script-intensive, manual, home-grown systems that
  - Are error prone
  - Do not scale well
  - Have little or no management visibility or reporting
- Multiple, disconnected build and test systems across locations, resulting in:
  - Redundant work
  - Inability to share or reuse code files across teams
  - Hard to manage build and test data
- Slow overall build and release cycles that directly impact:
  - Release predictability
  - Time-to-market

Automation Platform

The automation platform has a three-tier architecture, AJAX-powered web interface, and first-of-its-kind build and release analytic capabilities for reporting and compliance. With this solution, your developers, release engineers, build managers, QA teams, and managers gain:

- Shared platform for disseminating best practices and reusing common procedures
- Ability to support geographically distributed teams
- Continuous integration and greater agility
- Faster throughput and more efficient hardware utilization
- Visibility and reporting for better project predictability
- Better software quality by integrating and validating against all target platforms and configurations

For examples of ElectricFlow architecture configurations, see ElectricFlow Architecture on page 6.

What Makes ElectricFlow Unique?

ElectricFlow provides enterprise-class speed and scalability for software build and release management. It is easy to install and use on a simple build, yet scales to support the largest and most complex build and test processes. ElectricFlow distributes jobs in parallel across multiple resources for faster overall cycle time.

ElectricFlow supports multiple teams, working in multiple locations, programming in multiple languages in an environment that can be centrally controlled and managed. Shared assets and reuse make individual teams more efficient by eliminating duplicate work, and gives organizations the power to deploy cross-company standards.

ElectricFlow's unique analytics provide visibility into one of the best indicators of project success: compiled, tested, working code. ElectricFlow's analytics database stores all build and test information for real-time and trend reporting giving your organization the power to collect pinpoint statistics and to gain visibility into important productivity metrics such as trends in error rates. Additionally, out-of-the-box reports provide information about cross-project status and build trends by project and resource utilization. ElectricFlow's integration with virtual lab automation (VLA) solutions also allows you to snapshot or reproduce a specific build for auditing or troubleshooting purposes.

ElectricFlow provides unified process automation across the entire build-test-deploy life cycle and across heterogeneous tools via integrations with leading ALM tools. Integrations with SCM tools enable continuous integration, triggering builds whenever code is checked into the specified repository/branch. When used with VMware Lab Manager, ElectricFlow can dynamically provision either physical or virtual resources without manual intervention. This feature delivers efficient, dynamic resource provisioning and reduces development and QA dependence on IT operations.

ElectricFlow Architecture

ElectricFlow was designed to support small, mid-range, or enterprise scale software production. Based on a three-tier architecture, ElectricFlow scales to handle complex environments. The ElectricFlow multi-threaded Java server provides efficient synchronization even under high job volume.

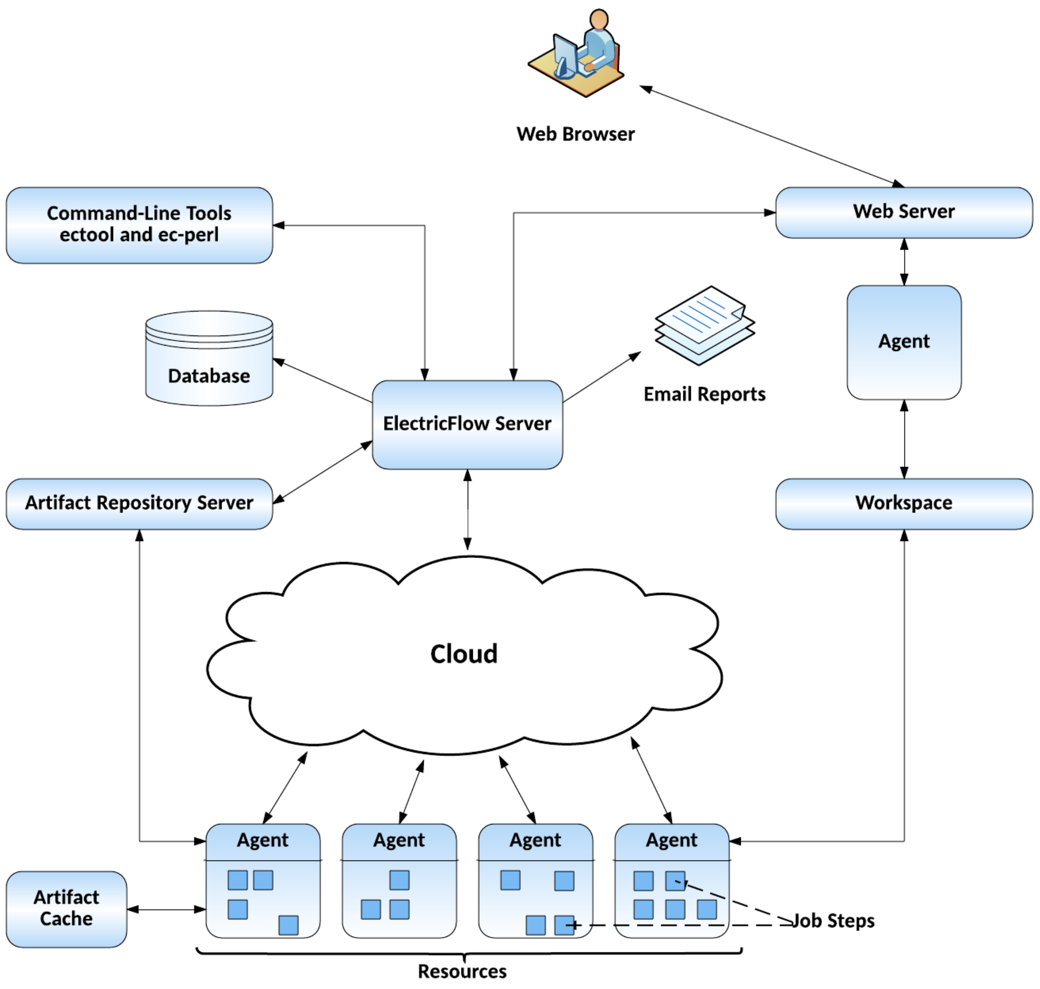
- The ElectricFlow server manages resources, issues commands, and generates reports.
- An underlying database stores commands, metadata, and log files.
- Agents execute commands, monitor status, and collect results in parallel across a cluster of servers for rapid throughput.

Simple Architectural Overview

This local configuration applies to all the use cases. The ElectricFlow server, web server, artifact cache, Artifact Repository server, workspace, command-line tools, resources, agents, and job steps are all in the automation platform.

In this local configuration:

- The ElectricFlow server manages resources, issues commands, and generates reports.
- Resources, agents, and databases are managed in the automation platform.
- An underlying database stores commands, metadata, and log files.
- Procedures, which include job steps, are defined in the automation platform.
- Job steps are executed on resources in the defined environments.
- Applications (which include procedures), components, and environments are defined for deployment automation.
- Pipelines, stages, and tasks are defined for pipeline management.



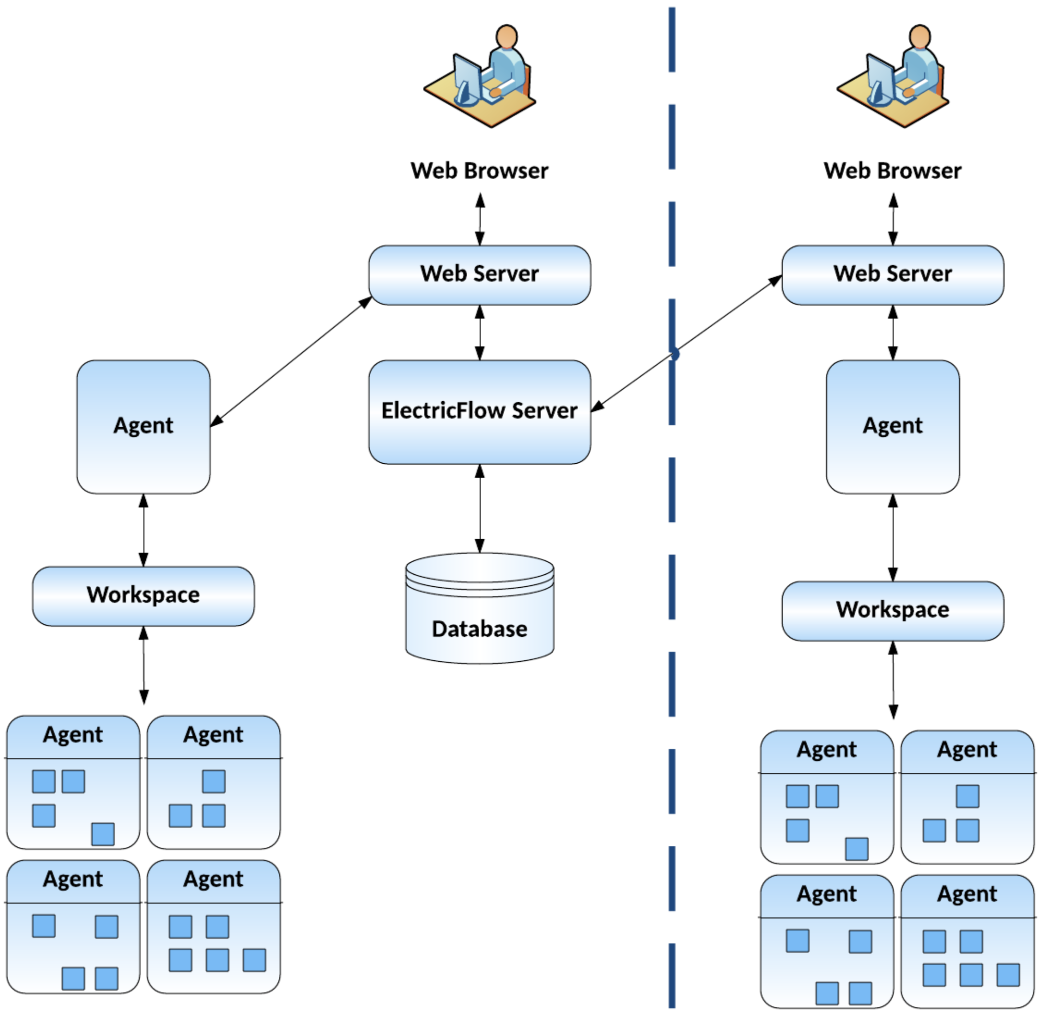
If you are only evaluating ElectricFlow, the ElectricFlow software, the database, the ElectricFlow server, the web server, and the repository server can reside on the same machine.

In a production environment, the database should reside on a separate machine from the ElectricFlow server to prevent performance issues. It is acceptable for the ElectricFlow server, web server, and repository server to reside on the same machine in a local configuration, but not required.

Expanded Remote Configuration

ElectricFlow is not limited by only the components shown in the previous configuration. This configuration applies to all the use cases.

The following shows a remote web server configuration and is an example for how you may set up a remote web server installation.



This type of remote web server configuration helps prevent network latency. If you have multiple sites, ElectricFlow can be configured to help you work more efficiently.

Other Configurations

Go to the ElectricFlow Installation Guide (http://docs.electric-cloud.com/eflow_doc/FlowIndex.html) for other architecture configurations:

- Proxy (universal) resources
- Remote database
- Multiple remote web servers
- Multiple remote repository servers
- Clustered configuration for horizontal scalability and high availability

Build-Test Automation

You create, configure, and manage these objects in the automation platform:

For build-test automation, you must create, configure, and manage these objects in the automation platform:

- Projects

  A *project* is an object used in ElectricFlow to organize information. A project is a container object for procedures, steps, schedules, workflows, and properties. If you use ElectricFlow for different purposes, you can use a separate project for each purpose so different projects do not interfere with each other. When you work in one project, you do not normally see information in other projects. At the same time, a project can use information defined in other projects, which allows you to create shared library projects.

- Resources

  A *resource* is defined as an agent machine where steps can execute. A resource has a logical name and a host name. In some situations, it is convenient to have multiple logical resources associated with the same host. A resource can also be associated with one or more pools. Each resource has a *step limit* that determines the maximum number of steps that can execute simultaneously on the resource. Resources can be grouped into *resource pools*. Multiple resources can be defined on the same machine.

- Procedures

  Procedures and *steps* define tasks that you want ElectricFlow to execute. A procedure consists of one or more steps. A step includes a command or script executed on a single resource and is the smallest unit of work that ElectricFlow understands. Each step specifies a resource on which it should run (either the name of a specific machine or the name of a *resource pool* of equivalent machines, in which case ElectricFlow picks a machine from the pool). A step can be given a time limit, and if the step does not complete within the specified time, ElectricFlow automatically aborts it.

  Steps are ordered within a procedure and normally execute sequentially. However, it is possible to mark a consecutive range of steps for parallel execution, so all steps in that range run concurrently.

  You can define *parameters* for procedures. Parameter values are assigned when procedures are scheduled. Parameters can be required, optional, or have default values. Parameters are used for a variety of purposes such as specifying the branch to build or the set of platforms on which to run tests. Parameter values can be used in step commands and many other places.

  Procedures can be nested. A step in one procedure can invoke another procedure instead of running a command. The invoking step provides parameters needed by the nested procedure, also referred to as a subprocedure.

- Schedules

  A schedule is used to execute procedures and determine when specific procedures run. A schedule can trigger at defined times, for example, every 2 hours from 10:00 pm to 6:00 am on Mondays, Wednesdays, and Fridays, or when modifications are checked into a particular branch of your source code control system. It is also possible to create a schedule that runs immediately and disappears after the job runs. When you create a schedule, you must provide the parameters required by the procedure that you want to invoke.

  The Continuous Integration Dashboard works with your source code management (SCM) system and provides visibility into running builds, the ability to add a project to continuous integration quickly, and easily accessed configuration pages to setup or modify a continuous integration schedule.

- Workflows

  Managing a build-test-deploy product life cycle spanning multiple procedures and projects requires a significant amount of "meta-programming" and a heavy use of properties, and the *workflow* feature simplifies this process. Using the workflow object, you can create build-test-deploy life cycles by defining a set of states and transitions. Any ElectricFlow project can contain a workflow.

- When a procedure is executed or run, a *job* is created. A job is an object that is created each time a procedure begins to execute or run. The job keeps track of all data associated with the procedure's execution, such as the running time of each step and any errors that may occur during the step. ElectricFlow retains job information after the job completes so you can examine what occurred.

After setting resources, procedures, and schedules, ElectricFlow automatically runs the procedures that you created using these objects and facilities:

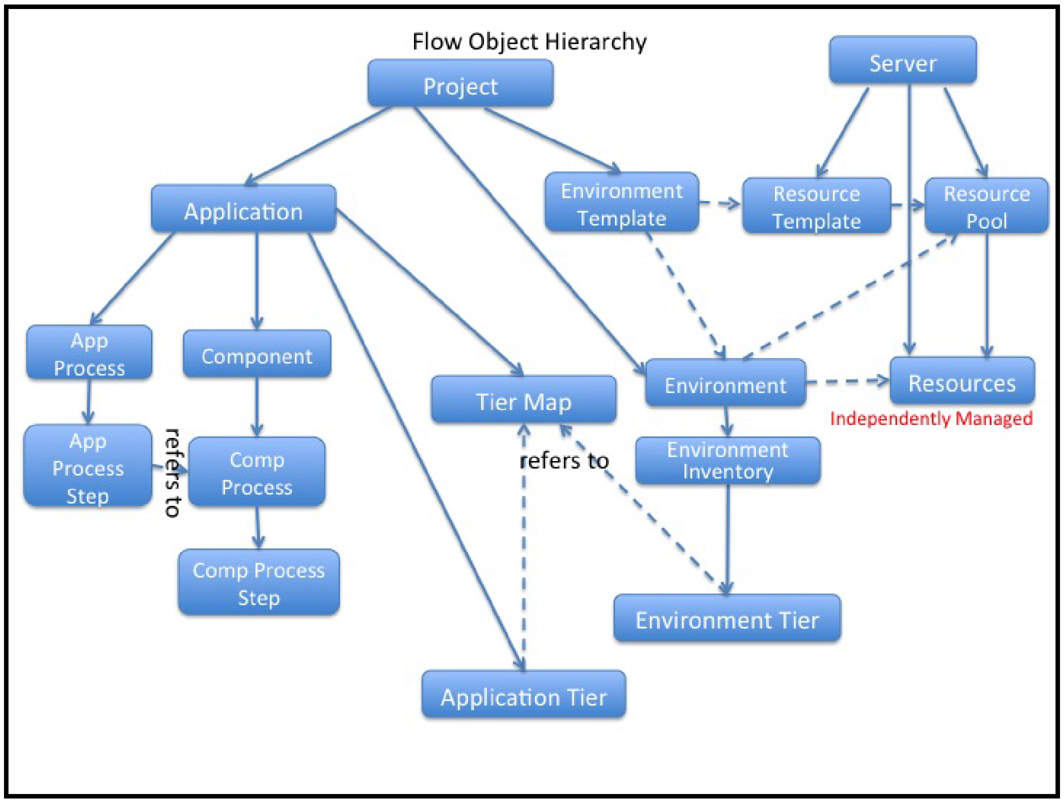
- Zones and Gateways–A zone (or top-level network) that you create is a way to partition a collection of agents to secure them from use by other groups. A gateway is a secured connection between two zones when you want to share or transfer information between the zones. For example, you might want a developers zone and a test zone. The ElectricFlow server is a member of the *default* zone, created during ElectricFlow installation.

- *Continuous Integration Builds* and other schedules–Run jobs according to *schedules* that you define. Scheduled jobs can run at specific times or when source code changes are checked in to your source control system. ElectricFlow integrates with major source control systems. The Continuous Integration Dashboard allows you to add more projects easily and create build configurations quickly so you can visually see running builds, build status, and so on.
- *Artifact Management* functionality–Using artifacts can improve performance across builds, provide better reusability of components, and improve cross-team collaboration with greater tractability. For example, instead of developers repeatedly downloading third-party packages from external sources, these components can be published and versioned as an artifact. Developers then simply retrieves a specific artifact version from a local repository, guaranteeing a consistent package from build to build.
- *Preflight build functionality*–Used by developers to build and test code changes in isolation on their local machines before those changes are committed to a production build.
- *Plugin* capability–ElectricFlow is built with an extensible UI, enabling easy development of plugins that include integrations with other tools, custom dashboards, and unique user experiences based on roles. "Bundled" plugins, installed during ElectricFlow installation, provide easy integration with your SCM systems, defect tracking applications, and so on. For a complete list of bundled plugins, see Appendix A: Plugins That are Bundled with ElectricFlow.
- *Workflow* functionality–Use a workflow to design and manage processes at a higher level than individual jobs. You can use workflows to combine procedures into processes to create build-test-deploy life cycles (for example). A workflow contains states and transitions that you define to provide complete control over your workflow process. The ElectricFlow Workflow feature allows you to define an unlimited range of large or small life cycle combinations to meet your needs.
- *Resource* management–If a resource is overcommitted, ElectricFlow delays some jobs until others are finished with the resource. You can define pools of equivalent resources, and ElectricFlow spreads usage across the pool.
- Recording a variety of information about each job, such as the running time and the success or failure of each step. A set of reports is available to provide even more information.
- Powerful and flexible *reporting* facilities–Various statistics such as number of compiles or test errors are collected after each step and recorded in the ElectricFlow database. A variety of reports can be generated from this information.
- Allowing you to observe jobs as they run and to cancel jobs.
- Credentials–Use a credential, consisting of an user name and password, can be attached directly to a step or schedule at the platform level. You can also attach impersonation credentials to procedure steps, procedures, and projects before executing the job step. It allows ElectricFlow to use a specific account with special privileges on a per-job or per-step basis.
- *Workspace* for each job, which is a disk area a job uses for storage–ElectricFlow also provides a facility for reclaiming space occupied by workspaces.
- Powerful data model based on *properties*–Properties are used to store job input data such as the source code branch to use for the build, to collect data during a job (such as number of errors or warnings), and to annotate the job after it completes (for example, a build has passed QA).
- *Access control* for users logged into the system–ElectricFlow uses this information to control their activities and integrates with Active Directory and LDAP repositories.
- *Search, sort, and filter* functions to minimize viewing or "wading" through information that is of no interest to you, allowing you to access to the information you need quickly.

- *Email notifications* to get important information or data to individuals or groups immediately and on a regular basis for a particular job or a specific job aspect.
- All ElectricFlow operations and features are available from a command-line application tool (Perl API), *ectool*, the RESTful API, DSL methods, and a user interface (UI).

This diagram shows the relationships between objects in the automation platform to objects used in deployment automation.

- Resources are assigned to environments.
- Resource pools are also assigned to environments.
- Resource pools are assigned to Resource Templates, which are used to define Environment Templates.



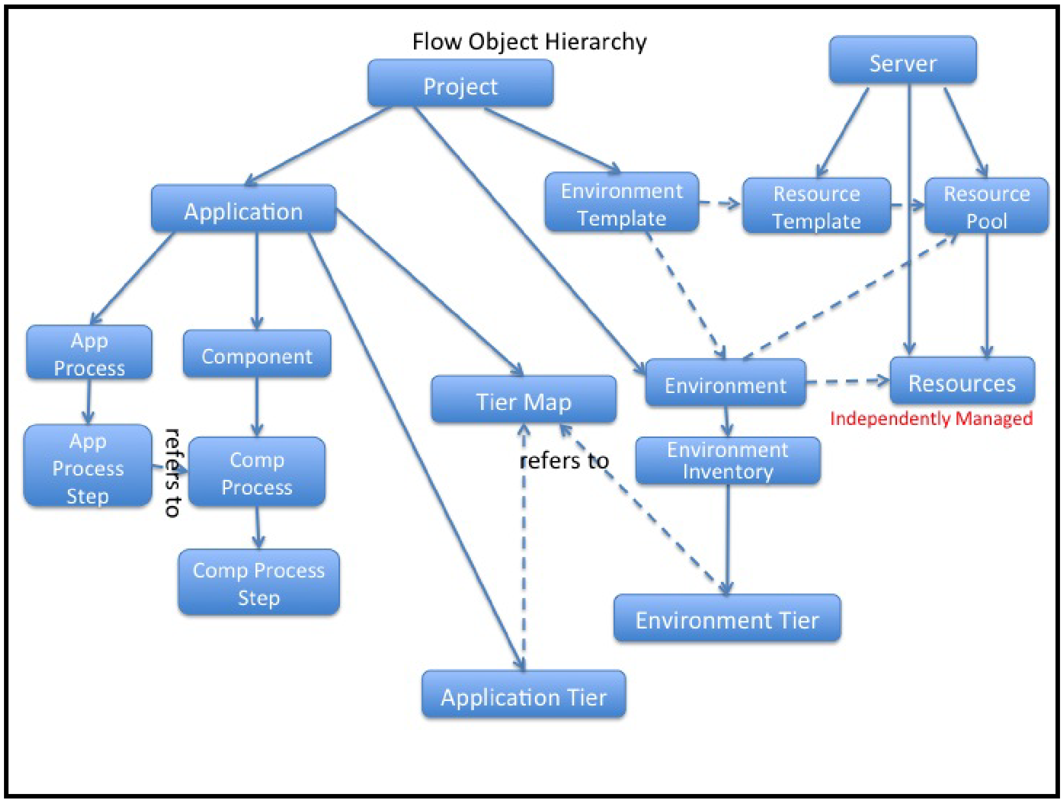
For more information about the ElectricFlow objects, concepts, and features in this topic, go to the ElectricFlow GlossaryGlossary on page 918

To configure and manage build-test automation, you can use API commands or DSL scripts.

You can also use the ElectricFlow user interface (UI) to configure and manage your automation solution. For information about using the ElectricFlow UI, see the ElectricFlow User Guide.

Deployment Automation

This diagram shows the relationships between the following objects to other objects in deployment automation.



It also shows the relationships to following objects in the automation platform:

- Resources are assigned to environments.
- Resource pools are also assigned to environments.
- Resource pools are assigned to Resource Templates, which are used to define Environment Templates.

To automate your deployments for Continuous Delivery, you model and deploy (run) applications in ElectricFlow.

- *Applications* consist of application processes and application tiers.

  You add components to application tiers and model component processes.

  Components are based on artifacts that are defined and managed by the automation platform.

- Before deploying an application, you map an application process to an environment, where the application will be deployed, in a *tier map*.

  A tier map can have one or more mappings of an application tier to an environment tier.

  An environment tier can be mapped to more than one application tier.

- *Environments* can be static or dynamic.

  You can create a *static environment* before deploying an application, or you can create a *dynamic environment* when deploying the application.

  An environment consists of one or more environment tiers to which resources are added.

  In a static environment, you can add only static resources to the environment tiers. These resources are defined and managed in the automation platform.

  You can create dynamic environments with provisioned cloud resources and static resources in ElectricFlow 5.4 or later.

Apply these features in your application:

- Dynamic environments

  A dynamic environment is automatically spun up on an on-demand basis when you deploy an application. It can have provisioned cloud resources and static resources.

  Dynamic environments allow you to optimize how your cloud resources are used, reuse provisioned resource pools, track the status and usage of cloud resources, and verify the credentials of these resources before provisioning them.

- Deploying applications

  You can deploy part or all of the objects one of these ways:

  - Full deploy–All objects in the application are deployed.
  - Smart deploy–Only objects that have not been deployed to specific resources, not deployed with specific artifact versions, or on new resources
  - Partial deploy–Only specific objects and versions
  - Schedule–On a one-time, daily, weekly, or monthly basis.
  - Snapshot–Based on a version of the application with specific artifact versions and the state of the application at any point in time.

    While developing an application, you can save different versions of the application as snapshots and compare them to refine and troubleshoot the application.

- Change Tracking

  ElectricFlow monitors changes to *tracked* objects, such as applications, procedures, workflows, workspaces, resources, and project-owned components (such as libraries). It records a *change history* of the historical states of the system and the state changes.

- Snapshots

  You can design and save a version of your application with specific artifact versions. If you save snapshots of the application during development and test phases, you can ensure that the components that were developed and tested are the same as those in the released version of the application. You can redeploy the snapshot any time.

- Credentials and impersonation

  You apply credentials and impersonation to control who can run applications and where the applications are run.

  - You can attach one or more credentials to component or application process steps.
  - You can attach only one impersonation credential to an application process, component process, or a process step.
  - When you attach an impersonation credential in ElectricFlow, it specifies the user who can deploy the application and the environment in which the application is deployed.
  - When you attach an impersonation credential in the automation platform, it specifies the account (user) that can run the job or job step. If you want to specify another condition, you have to attach another credential to the object.

- Custom parameters in application processes

  You can define and apply custom parameters to application processes in your deployments.

  You define the parameters and apply them while deploying the application or while defining an application process step, which determines when and how the application is deployed.

- Email notifications

  You can easily customize the email notification that the system sends when an application, application process, or process step runs.

  When setting the recipients of email notifications, you can specify users or groups, which are defined and managed in the automation platform, as well as email addresses.

- Tracking, viewing, and troubleshooting the deployment results

  Use the Environment Inventory to track and view details of the objects that were deployed and artifacts in the application. It shows the status of the application deployment at a point in time.

  Use the Application Inventory to track and view the deployment results. It shows more details about the application at a point in time.

  You can also view the change history of the objects in the application and search for specific information.

More about application, deploy, and run:

As you use ElectricFlow, remember that these terms have different meanings within ElectricFlow *and* outside of ElectricFlow when you deploy your software or application:

Term	Within ElectricFlow	Outside of ElectricFlow
Application	The application that you design and run (deploy) to produce your software for continuous delivery across different pipelines.	The software, system or application that you build, test, install, implement, release, and deploy using ElectricFlow. This is the end product of using ElectricFlow.
Deploy	Running the application that you designed in ElectricFlow. The end product is your software, system, or application. Deploy is a synonym of run in ElectricFlow.	All the processes or actions to develop and run your software in its environment, including building, testing, implementing, installing, configuring, making changes, and releasing.
Run	Running the application that you designed. The end product is your software, system, or application. *Run* is a synonym of *deploy* in ElectricFlow.	All the processes or actions to use software in its environment, including implementing, installing, configuring, debugging, troubleshooting, and releasing.

For more information about the ElectricFlow objects, concepts, and features in this topic, go to the ElectricFlow GlossaryGlossary on page 918

To configure and manage deployment automation, you can use API commands or DSL scripts.

You can also use the ElectricFlow user interface (UI) to configure and manage your automation solution. For information about using the ElectricFlow UI, see the ElectricFlow User Guide.

Pipeline Management

For end-to-end Continuous Delivery, you model and deploy pipelines in ElectricFlow.

... concepts, features, objects

For more information about the ElectricFlow objects, concepts, and features in this topic, go to the ElectricFlow GlossaryGlossary on page 918

To configure and manage pipeline management, you can use API commands or DSL scripts.

You can also use the ElectricFlow user interface (UI) to configure and manage your automation solution. For information about using the ElectricFlow UI, see the ElectricFlow User Guide.

Roadmap to the ElectricFlow APIs

ElectricFlow supports these APIs, ranked from easiest to hardest to use:

- DSL methods

  You create scripts and templates without using API commands.

The ElectricFlow DSL allows you to create scripts or templates for all the operations that you can do on the ElectricFlow UI, using the RESTful API, or the Perl API.

- RESTful APIs

  You do not need detailed knowledge of the API syntax to execute RESTful API requests.

  You navigate to the RESTful API URI and enter the appropriate information in the API UI to execute a request.

- Perl API

  You need to know the correct syntax to execute these commands.

  You can use Perl APIs one of these ways:

  - Access ectool or ec-perl through the command-line interface
  - Put the API commands in Javascript

Go to the following sections to use these APIs:

API Type	**Go to**
DSL Methods	Using the ElectricFlow DSL on page 832
RESTful APIs	Using the ElectricFlow RESTful API on page 823
Perl APIs	Using ectool on page 19 Using Perl (ec-perl) on page 20 Using API Commands in Javascript on page 28

Using the ElectricFlow Perl API

The Perl API is the most difficult of the ElectricFlow APIs to use because you need to know the command syntax to perform ElectricFlow operations such as

- Create and call procedures
- Model and deploy applications
- Create and manage resources
- Create environment models and add resources to them
- Model and run pipelines
- Model and run releases
- Create and manage artifacts
- Create and manage object properties

You can access the Perl API for ElectricFlow features one of these ways:

- Through the user interface (UI)

  The most common way is through the user interface (UI), also referred to as the *web interface* in this document.

  The UI displays windows and dialog boxes from which you can perform the following operations:

  - Create projects, procedures, and steps.
  - Launch jobs.
  - Deploy applications.
  - Manage all administration tasks at the automation-platform level.

- Through ectool or ec-perl

  The Perl APIs can be used from a command-line interface, in a shell script, or in a batch file.

  Any operation you can perform on the web interface, you can perform using the API because they all rely on the same interface to the ElectricFlow server.

  The ElectricFlow 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.

- Through Javascript

  The Perl APIs can be included in Javascript files.

  Any operation you can perform on the web interface, you can perform using Javascript files containing Perl APIs because they both rely on the same interface to the ElectricFlow server.

This topic describes ectool and ec-perl usage and their differences because ectool and ec-perl can work together. This topic also describe Javascript usage.

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 ElectricFlow 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 log in 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 an 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 (ec-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 ElectricFlow Perl API. ElectricFlow 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 ElectricFlow copy of Perl, passing all of its parameters to Perl.

To run `ec-perl` from a command line (or in an ElectricFlow step) 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 ElectricFlow-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 ElectricFlow 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 ElectricFlow package sets the server name based on the environment variable, COMMANDER_SERVER, set by the ElectricFlow 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 ElectricFlow 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`, and 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 ElectricFlow returned 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 ElectricFlow 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.

Global Arguments	Description
`--server <hostname>`	ElectricFlow 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, we recommend 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:** we recommend 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 ElectricFlow server. Defaults to port 8000.
`--securePort <secureport>`	HTTPS listener port on the ElectricFlow server. Defaults to port 8443.
`--secure <0|1>`	Use HTTPS to communicate with the ElectricFlow 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`.
`--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 ElectricFlow’s automatic error recovery. When the API is unable to contact the ElectricFlow 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.

Global Arguments	Description
`--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`, enter 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 enter 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 enter 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 ElectricFlow Perl Modules into Your Perl Distribution

You may want to use your existing Perl distribution. If so, ElectricFlow 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 ElectricFlow 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 ElectricFlow Perl Distribution

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

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

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

For Linux

From the command line use: `<ElectricFlow_Dir>/perl/bin/perl -MCPAN -e 'install <module>'`

For Windows

Compatibility with ElectricFlow is important. ElectricCommander 4.1 and later 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 ElectricFlow).

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, ElectricFlow 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 `<ElectricFlow_Dir>\perl\lib\config.pm` file to eliminate spaces in references to the ElectricFlow 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\li
  b',
    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.

Using API Commands in Javascript

These are examples of how to use Perl API commands in Javascript:

- To create a project:

```
ectool evalScript --value "(api.createProject( { 'projectName':'alex34' } )).pro
ject.projectName ; " alex34
```

- To return the object type, use this Javascript API:

```
ectool evalScript --value "api.getResources({})" [object Object]
```

- For a parsed object, use the `"JSON.stringify()"` call:

```
ectool evalScript --value "JSON.stringify(api.getResources({})) "{"resource":{"r
esourceId":"ceecfce5-2d0d-11e4-8888-005056330afe","resourceName":"local","agentS
tate":{"alive":"1","details":"The agent is alive","hostOS":"Linux qa-ub1210-64-2
3.5.0-19-generic #30-Ubuntu SMP Tue Nov 13 17:48:01 UTC 2012 x86_64 x86_64 x86_6
4 GNU/Linux","hostPlatform":"linux","message":"The agent is alive","pingToken":"
1409049660","protocolVersion":"6","state":"alive","time":"2014-08-26T10:43:25.80
2Z","version":"5.0.3.76444"},"createTime":"2014-08-26T10:43:25.617Z","descriptio
n":"Local resource created during installation.","hostName":"qa-ub1210-64-2.elec
tric-cloud.com","hostOS":"Linux qa-ub1210-64-2 3.5.0-19-generic #30-Ubuntu SMP T
ue Nov 13 17:48:01 UTC 2012 x86_64 x86_64 x86_64 GNU/Linux","hostPlatform":"linu
x","lastModifiedBy":"project: zebra","lastRunTime":"2014-08-26T10:50:23.786Z","m
odifyTime":"2014-08-26T10:50:23.786Z","owner":"admin","port":"7800","proxyPor
t":"",
"resourceAgentState":"alive","resourceAgentVersion":"5.0.3.76444",
"resourceDisabled":"0","stepCount":"0","stepLimit":"","trusted":"0",
"useSSL":"1","propertySheetId":"ceee8387-2d0d-11e4-8888-005056330afe","zoneName"
:"default","pools":"default"}}
```

- To get the first `resourceName`:

```
ectool evalScript --value "api.getResources({}).resource[0].resourceName"
```

ElectricFlow Perl API Commands

Click the following link to view the entire command list by group. In the UI, you can use the Help system Search feature to quickly locate the Perl API commands.

- Perl API Commands Listed by Group

  Commands are grouped into common usage sections for your convenience.
  This view is helpful if you want to see all available commands for a particular object.

Click a API command name to go to a section for that API command containing arguments and their descriptions, command syntax, and usage examples.

Note: The API tables display positional arguments for each command; however, you can use "value pairs" to construct your command scripts instead. For more information, see the "Using the ElectricFlow Perl API on page 18" topic.

Perl API Commands Listed by Group

The ElectricFlow API commands in the following tables are listed in alphabetical order in each group.

Click a command name to go to the section with expanded information for that command, including its arguments (required and optional), descriptions, usage examples, and related commands.

ACL Management (Access Control List)

Commands	Description
`breakAclInheritance`	Breaks ACL (access control list) inheritance at the given object.
`checkAccess`	Checks ACL (access control list) permission information associated with an object (including inherited ACLs) for the current user.
`createAclEntry`	Creates an ACE (access control list entry) on an object for a given principal.
`deleteAclEntry`	Deletes an ACE on an object for a given principal.
`getAccess`	Retrieves ACL information associated with an object, including inherited ACLs.
`getAclEntry`	Retrieves an ACE on an object for a given principal.
`modifyAclEntry`	Modifies an ACE on an object for a given principal.
`restoreAclInheritance`	Restores ACL inheritance at the given object.

Applications

Commands	Description
createApplication	Creates a new application for a project.
deleteApplication	Deletes an application.
getApplication	Retrieves an application by name.
getApplications	Retrieves all applications in a project.
modifyApplication	Modifies an existing application.

Application Tiers

Commands	Description
createApplicationTier	Creates a new application tier in the application.
deleteApplicationTier	Deletes a tier from an application.
getApplicationTier	Retrieves an application tier by name
getApplicationTiers	Retrieves all application tiers in an application
getApplicationTiersInComponent	Retrieves all application tiers that are used by the given component
modifyApplicationTier	Modifies an existing tier in the application.

Artifact Management

Commands	Description
addDependentsToArtifactVersion	Adds an artifact version query to an existing artifact. Dependent artifact versions are retrieved when the parent artifact version is retrieved.
cleanupArtifactCache	Deletes stale artifact versions from an artifact cache. A "stale artifact version" is one whose metadata was previously deleted from the ElectricFlow server.
cleanupRepository	Deletes stale artifact versions from the repository backing-store. A "stale artifact version" is one whose metadata was previously deleted from the ElectricFlow server.

Commands	Description
`createArtifact`	Creates a new artifact.
`createArtifactVersion`	Creates a new artifact version.
`createRepository`	Creates a repository for one or more artifacts.
`deleteArtifact`	Deletes an existing artifact element and all artifact versions.
`deleteArtifactVersion`	Deletes artifact version metadata from the ElectricFlow database. (This API call does not delete or remove artifacts stored on the repository machine.)
`deleteRepository`	Deletes artifact repository metadata from the ElectricFlow database. (This API call does not delete or remove artifacts stored on the repository machine.)
`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. Use the Perl API for the `findArtifactVersions` command.
`getArtifact`	Retrieves an artifact by its name.
`getArtifacts`	Retrieves all artifacts in the system.
`getArtifactVersion`	Retrieves an artifact version by its name.
`getArtifactVersions`	Retrieves all artifact versions in the system, filtered by artifact name, retriever job ID, and/or retriever job step ID.
`getManifest`	Retrieves the manifest for a specified artifact version, which includes a list of files and directories in the artifact version, plus its checksum file.
`getRepositories`	Retrieves all artifact repository objects known to the ElectricFlow server.
`getRepository`	Retrieves an artifact repository by its name.
`getRetrievedArtifacts`	Retrieves artifacts during a job.
`modifyArtifact`	Modifies an existing artifact.
`modifyArtifactVersion`	Modifies an existing artifact version.
`modifyRepository`	Modifies an existing artifact repository.

Commands	Description
moveRepository	Moves an artifact repository in front of another, specified repository or to the end of the list.
publishArtifactVersion	Publishes an artifact version to an artifact repository.
removeDependentsFromArtifactVersion	Removes a list of dependent artifact versions from an existing artifact version.
resolveRoute	Resolves the network route to an artifact repository.
retrieveArtifactVersions	Retrieves the most recent artifact version, including its dependents, from an artifact repository.
updateArtifactVersion	Updates an artifact version by adding or replacing one or more files in the existing file and publishes the result as a new artifact version to an artifact repository.

Change History

Commands	Description
getDeploymentHistoryItems	Retrieves all the deployment history items for a specific environment.
getEntityChange	Retrieves the entity changes.
getEntityChangeDetails	Retrieves the differences between entities.
pruneChangeHistory	Prunes obsolete-for-days data from the Change History tables.
revert	Reverts the state of the object to a previous state.
searchEntityChange	Searches for entity changes.

Components

Commands	Description
addComponentToApplicationTier	Adds the specified component to the specified application tier.
copyComponent	Creates a new component based on an existing one.
createComponent	Creates a new component for a project.

Commands	Description
deleteComponent	Deletes a component.
getComponent	Retrieves a component by name
getComponents	Retrieves all components in a project
getComponentsinApplicationTier	Retrieves the list of components in an application tier.
modifyComponent	Modifies an existing component.
removeComponentFromApplicationTier	Removes the specified component from an application tier.

Credential Management

Commands	Description
attachCredential	Attaches a credential to an object.
createCredential	Creates a new credential for a project.
deleteCredential	Deletes a credential.
detachCredential	Detaches a credential from an object.
getCredential	Finds a credential by name.
getCredentials	Retrieves all credentials in a project.
getFullCredential	Finds a credential by name, including password, from within a running step.
modifyCredential	Modifies an existing credential.

Database Configuration

Commands	Description
getDatabaseConfiguration	Retrieves the current database configuration.

Commands	Description
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 running, these changes have no effect until the server is restarted.
validateDatabase	Performs consistency checks on the database.

Directory Provider Management

Commands	Description
createDirectoryProvider	Creates a new LDAP directory provider.
deleteDirectoryProvider	Deletes an LDAP directory provider.
getDirectoryProvider	Retrieves an LDAP directory provider by name.
getDirectoryProviders	Retrieves all LDAP directory providers.
modifyDirectoryProvider	Modifies an existing LDAP directory provider.
moveDirectoryProvider	Moves an LDAP directory provider in front of another specified provider or to the end of the list.
testDirectoryProvider	Tests an LDAP directory provider.

Dynamic Environments

Commands	Description
addResourcePoolToEnvironmentTier	Adds a resource pool to a specific environment tier.
addResourceTemplateToEnvironmentTemplateTier	Adds a resource template to the specified environment template tier.
addResourceToEnvironmentTemplateTier	Adds a resource to the specified environment template tier.
createEnvironmentTemplate	Creates an environment template.
createEnvironmentTemplateTier	Creates a tier in an environment template.

Commands	Description
createEnvironmentTemplateTierMap	Creates an environment-template tier map for an application.
createHook	Creates a hook in a resource template, which can have one or more hooks.
createResourceTemplate	Creates a resource template.
deleteEnvironmentTemplate	Deletes an environment template.
deleteEnvironmentTemplateTier	Deletes an environment template tier.
deleteEnvironmentTemplateTierMap	Deletes an environment template tier map from an application.
deleteEnvironmentTemplateTierMapping	Deletes a tier mapping from a environment-template tier map.
deleteHook	Deletes a hook associated with an entity.
deleteResourceTemplate	Deletes a resource template.
getAvailableResourcesForEnvironment	Retrieve all non-dynamic resources or resource pools.
getEnvironmentTemplate	Retrieves an environment template.
getEnvironmentTemplateTier	Retrieves an environment tier in an environment template.
getEnvironmentTemplateTierMaps	Retrieves all the environment-template tier maps used by the specified application.
getEnvironmentTemplateTiers	Retrieves all the environment template tiers in the specified environment template.
getEnvironmentTemplates	Retrieves all the environment templates in the specified project.
getHook	Retrieves a hook associated in an entity.
getHooks	Retrieves all the hooks associated with an entity.
getProvisionedEnvironments	Retrieves provisioned environments.
getResourcePoolsInEnvironmentTier	Retrieves the list of resource pools in the specified environment tier.
getResourceTemplate	Retrieves the specified resource template.
getResourceTemplates	Retrieves all the resource templates.

Commands	Description
getResourceTemplatesInEnvironmentTemplateTier	Retrieves all the resource templates in the specified environment template tier.
getResourcesInEnvironmentTemplateTier	Retrieves all the resources in the specified environment template tier.
modifyEnvironmentTemplate	Modifies an environment template.
modifyEnvironmentTemplateTier	Modifies all the environment template tiers in the specified environment template.
modifyEnvironmentTemplateTierMap	Modifies an existing environment template tier map.
modifyEnvTemplTierResourceTemplMapping	Modifies the resource count in an environment template tier.
modifyHook	Modifies an existing hook in a resource template.
modifyResourceTemplate	Modifies the specified resource template.
provisionEnvironment	Provisions an environment.
provisionResourcePool	Provisions a resource pool.
removeResourceFromEnvironmentTemplateTier	Removes a resource from an environment template tier.
removeResourcePoolFromEnvironmentTier	Removes a resource pool from the specified environment tier.
removeResourceTemplateFromEnvironmentTemplateTier	Removes a resource template from the specified environment template tier.
tearDown	Removes dynamic environments that are no longer needed.

Email Configuration and Management

Commands	Description
createEmailConfig	Creates a new email configuration.
deleteEmailConfig	Deletes an email configuration.
getEmailConfig	Retrieves an email configuration by name.

Commands	Description
getEmailConfigs	Retrieves all email configurations.
modifyEmailConfig	Modifies an existing email configuration.

Email Notifiers Management

Commands	Description
createEmailNotifier	Creates an email notifier on an object specified by an `emailNotifierSelector`.
createEventSubscription	Creates a list of event subscriptions.
deleteEmailNotifier	Deletes an email notifier from a property sheet container.
deleteEventSubscription	Deletes a list of event subscriptions.
getEmailNotifier	Retrieves an email notifier from a property sheet container.
getEmailNotifiers	Retrieves all email notifiers defined for the specified property sheet container.
modifyEmailNotifier	Modifies an email notifier in a property sheet container specified by an `emailNotifierSelector`.
modifyEventSubscription	Modifies a list of event subscriptions.
sendEmail	Facilitates sending an email from the command-line or a Command 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.

Environments

Commands	Description
createEnvironment	Creates a new environment.
createEnvironmentInventoryItem	Creates a new environment inventory item.
createReservation	Creates a new reservation.

Commands	Description
deleteEnvironment	Deletes an environment.
deleteEnvironmentInventoryItem	Deletes an inventory item.
deleteReservation	Deletes a reservation.
getEnvironment	Retrieves an environment by name.
getEnvironmentApplications	Retrieves a list of applications installed on the given environment.
getEnvironmentInventory	Retrieves a per-component grouped list of inventory items.
getEnvironmentInventoryItem	Retrieves an inventory item.
getEnvironmentInventoryItems	Retrieves all the inventory items for a given environment.
getEnvironments	Retrieves all environments in a project.
getReservation	Retrieves an environment reservation by its name.
getReservations	Retrieves all the environment reservations.
getRunSchedules	Retrieves the run schedules with environment reservations
modifyEnvironment	Modifies an environment.
modifyEnvironmentInventoryItem	Modifies an existing environment inventory item.
modifyReservation	Modifies an environment reservation.
seedEnvironmentInventory	Creates a new environment.

Environment Tiers

Commands	Description
addResourcesToEnvironmentTier	Adds resources to the specified environment tier.
createEnvironmentTier	Creates a new environment tier.
deleteEnvironmentTier	Deletes an environment tier.
getEnvironmentTier	Retrieves an environment tier by name.

Commands	Description
getEnvironmentTiers	Retrieves all environment tiers in an environment.
modifyEnvironmentTier	Modifies an environment tier.
removeResourcesFromEnvironmentTier	Removes the given resources from the given environment tier.

Gateway and Zone Management

Commands	Description
createGateway	Creates a new gateway.
deleteGateway	Deletes a gateway.
getGateway	Finds a gateway by name.
getGateways	Retrieves all gateways.
modifyGateway	Modifies an existing gateway.
createZone	Creates a new zone.
deleteZone	Deletes a zone.
getZone	Finds a zone by name.
getZones	Retrieves all zones.
modifyZone	Modifies an existing zone.

Job Management

Commands	Description
abortAllJobs	Aborts all running jobs.
abortJob	Aborts a running job.
abortJobStep	Aborts any type of step—command step or subprocedure step.
completeJob	Completes an externally managed job.

Commands	Description
`completeJobStep`	Completes an externally managed job step.
`createJob`	Creates an externally managed job.
`createJobStep`	Creates a job step in an existing job.
`deleteJob`	Deletes a job from the ElectricFlow database.
`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 ElectricFlow UI.
`getJobDetails`	Retrieves complete information about a job, including details from each job step.
`getJobInfo`	Retrieves all information about a job, except job step information.
`getJobNotes`	Retrieves the notes property sheet from a job.
`getJobs`	Retrieves summary information for a list of jobs.
`getJobsForSchedule`	Retrieves jobs started by a specific schedule.
`getJobStatus`	Retrieves the status of a job.
`getJobStepDetails`	Retrieves details for a job step.
`getJobStepStatus`	Retrieves the status of a job step.
`getJobSummaries`	Retrieves summary information about jobs.
`getJobSummary`	Retrieves a job and its job steps with only the specified job and job step properties.
`modifyJob`	Modifies the status of an externally managed job.
`modifyJobStep`	Modifies the status of an externally managed job step.
`moveJobs`	Moves jobs from one project to another.
`runProcedure`	Creates and starts a new job using a procedure directly or specified indirectly through a schedule.
`setJobName`	Sets the name of a running job.
`waitForJob`	Waits until the specified job reaches a given status or the timeout expires

Parameter Management

Commands	Description
`attachParameter`	Attaches a formal parameter to a step.
`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 runtime, the actual parameter name needs to match the name of a formal parameter in the nested procedure.
`createFormalParameter`	Creates a new formal parameter for a procedure.
`deleteActualParameter`	Deletes an actual parameter.
`deleteFormalParameter`	Deletes a formal parameter.
`detachParameter`	Detaches a formal parameter from a step.
`getActualParameter`	Retrieves an actual parameter by its name.
`getActualParameters`	Retrieves all actual parameters from a job, job step, or step.
`getFormalParameter`	Retrieves a formal parameter by its name.
`getFormalParameterOptions`	Retrieves possible option values for a procedure's formal parameter in the procedure using the options script registered for it.
`getFormalParameters`	Retrieves all formal parameters from a procedure, schedule, or step.
`modifyActualParameter`	Modifies an existing actual parameter. An actual parameter is a name/value pair that is passed to a subprocedure. This command supports renaming the actual parameter and setting its value.
`modifyFormalParameter`	Modifies an existing formal parameter.
`validateFormalParameters`	Validates input parameters for a procedure using the validation script registered for it.

Pipelines and Releases

Commands	Description
`abortAllPipelineRuns`	Aborts all pipeline runs associated with a release..
`abortPipelineRun`	Aborts a pipeline run.

Commands	Description
`completeManualTask`	Completes the manual task.
`completeRelease`	Completes the Release.
`createDeployer`	Creates a new Deployer task for a project or a Release.
`createDeployerApplication`	Creates a Deployer application for a Release.
`createDeployerConfiguration`	Creates the Deployer configuration to the Deployer application.
`createGate`	Creates a new gate for a stage.
`createNote`	Creates a new note.
`createPipeline`	Creates a new pipeline for a project.
`createRelease`	Creates a new Release for a project.
`createStage`	Creates a new stage in a pipeline for a project.
`createTask`	Creates a new task for a task container.
`deleteGate`	Deletes a gate for a stage.
`deleteNote`	Deletes a note associated with an entity.
`deletePipeline`	Deletes a pipeline in a project.
`deletePipelineRun`	Deletes a pipeline runtime.
`deleteRelease`	Deletes a Release.
`deleteStage`	Deletes a stage in a project.
`deleteTask`	Deletes a task in a task container.
`getDeployerApplication`	Retrieves the application used in the Release by name.
`getDeployerApplications`	Retrieves all the applications in a Release.
`getDeployerConfiguration`	Retrieves a Deployer configuration.
`getDeployerConfigurations`	Retrieves all the configurations in the Deployer.
`getGate`	Retrieves a gate by its stage name and gate type.
`getNote`	Retrieves a note associated with an entity.

Commands	Description
getNotes	Retrieves all the notes associated with an entity.
getPipeline	Retrieves a pipeline by its name.
getPipelineRuntimeDetails	Retrieves pipeline runtime details.
getPipelineRuntimes	Retrieves pipeline runs
getPipelineStageRuntimeDeployerTasks	Returns the list of Deployer tasks and their details to be displayed in Pipeline Run Details page.
getPipelineStageRuntimeTasks	Retrieves the list of pipeline stage tasks and the details about them that are displayed in the pipeline run view.
getPipelines	Retrieves all the pipelines.
getRelease	Retrieves a Release by name.
getReleaseInventory	Retrieves inventory artifacts created in a Release.
getReleases	Retrieves all releases.
getStage	Retrieves a stage by its name.
getStages	Retrieves all the stages for a pipeline.
getTask	Retrieves a task by its name.
getTasks	Retrieves a task by name.
getWaitingTasks	Retrieves a list of all the stage tasks that are currently waiting on manual tasks in pipeline run view.
modifyDeployer	Modifies an existing Deployer.
modifyDeployerApplication	Modifies the Deployer application associated with a Release.
modifyDeployerConfiguration	Modifies a Deployer configuration associated with a Deployer application.
modifyGate	Modifies an existing gate.
modifyNote	Modifies a note associated with an entity.
modifyPipeline	Modifies an existing pipeline.
modifyRelease	Modifies an existing Release.

Commands	Description
modifyStage	Modifies an existing stage.
modifyTask	Modifies an existing task.
removeDeployerApplication	Removes a Deployer application for a Release.
removeDeployerConfiguration	Removes a Deployer configuration associated with a Deployer application.
runPipeline	Runs the specified pipeline.
startRelease	Starts a Release.
validateDeployer	Validates the Deployer configuration.
waitForFlowRuntime	Waits until the pipeline specified by the flow runtime ID is completed or the timeout expires.

Plugin Management

Commands	Description
createPlugin	Creates a plugin from an existing project.
deletePlugin	Deletes an existing plugin object without deleting the associated project or files.
getPlugin	Retrieves an installed plugin.
getPlugins	Retrieves all installed plugins.
installPlugin	Installs a plugin from a JAR file. Extracts the JAR contents on the server and creates a project and a plugin.
modifyPlugin	Modifies a plugin.
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.
uninstallPlugin	Uninstalls a plugin, deleting the associated project and any installed files.

Procedure Management

Commands	Description
`createProcedure`	Creates a new procedure for an existing project.
`createStep`	Creates a new procedure step.
`deleteProcedure`	Deletes a procedure, including all steps.
`deleteStep`	Deletes a step from a procedure.
`getProcedure`	Finds a procedure by its name.
`getProcedures`	Retrieves all procedures in a project.
`getStep`	Retrieves a step from a procedure.
`getSteps`	Retrieves all steps in a procedure.
`modifyProcedure`	Modifies an existing procedure.
`modifyStep`	Modifies an existing step.
`moveStep`	Moves a step within a procedure.

Process

Commands	Description
`createProcess`	Creates a new process for an application or component.
`deleteProcess`	Deletes an application or component process.
`getProcess`	Retrieves an application or component process.
`getProcesses`	Retrieves all processes in an application or component.
`modifyProcess`	Modifies an existing process.
`runProcess`	Runs the specified process.

Process Dependency Management

Commands	Description
createProcessDependency	Creates a dependency between two process steps.
deleteProcessDependency	Deletes a dependency between two process steps.
getProcessDependencies	Retrieves all dependencies for a process.
modifyProcessDependency	Modifies a dependency between two process steps.

Process Step Management

Commands	Description
createProcessStep	Creates a new process step.
deleteProcessStep	Deletes an application or component process step.
getProcessStep	Retrieves an application or component process step.
getProcessSteps	Retrieves all the process steps in an application or component process.
modifyProcessStep	Modifies an existing process step.

Project Management

Commands	Description
createProject	Creates a new project.
deleteProject	Deletes a project, including all procedures, procedure steps, and jobs.
getProject	Finds a project by its name.
getProjects	Retrieves all projects.
modifyProject	Modifies an existing project.
reloadSetupScripts	Runs new, modified, or previously unsuccessful setup scripts.

Property Management

Commands	Description
`createProperty`	Creates a regular string or nested property sheet using a combination of property path and context.
`deleteProperty`	Deletes a property from a property sheet.
`expandString`	Expands property references in a string, in the current context.
`getProperties`	Retrieves all properties associated with an object.
`getProperty`	Retrieves the specified property value.
`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.
`modifyProperty`	Modifies a regular string or nested property sheet using a combination of property path and context.
`setProperty`	Sets the value for the specified property.

Rolling Deployments

Commands	Description
`createRollingDeployPhase`	Adds a rolling deploy phase to the specified environment.
`deleteRollingDeployPhase`	Deletes the rolling deploy phase associated with an environment.
`getRollingDeployPhase`	Retrieves the rolling deploy phase associated with an environment.
`getRollingDeployPhases`	Retrieves all the rolling deploy phases associated with an environment.
`modifyRollingDeployPhase`	Modifies the rolling deploy phase associated with an environment.
`setTierResourcePhase`	Maps a resource to a rolling deploy phase.

Resource Management

Commands	Description
`addResourcesToPool`	Adds resources to a specified resource pool.
`addResourceToEnvironmentTier`	Adds a resource to the specified environment tier.
`createResource`	Creates a new resource.
`createResourcePool`	Creates a pool container for resource.
`deleteResource`	Deletes a resource.
`deleteResourcePool`	Deletes a resource pool.
`getResource`	Retrieves a resource by its name.
`getResources`	Retrieves all resources.
`getResourcesInEnvironmentTier`	Retrieves the list of resources in an environment tier.
`getResourcesInPool`	Retrieves a list of resources in a pool.
`getResourcePool`	Retrieves a specified resource pool by name.
`getResourcePools`	Retrieves a list of resource pools.
`getResourceUsage`	Retrieves resource usage information.
`modifyResource`	Modifies an existing resource.
`modifyResourcePool`	Modifies an existing resource pool.
`pingAllResources`	Pings all resources.
`pingResource`	Pings one resource.
`removeResourceFromEnvironmentTier`	Removes a resource from the specified environment tier.
`removeResourcesFromPool`	Removes resources from a specified resource pool.
`runDiscovery`	Runs a plugin Discover procedure to discover contents of a list or set of resources, and store settings for them in the `ec_discovery` property sheets.
`signCertificate`	Signs an agent certificate.

Schedule Management

Commands	Description
createSchedule	Creates a new schedule.
deleteSchedule	Deletes a schedule.
getSchedule	Retrieves a schedule by its name.
getSchedules	Retrieves all schedules.
modifySchedule	Modifies an existing schedule.
pauseScheduler	Sets the scheduler to pause.

Server Management

Commands	Description
deleteLicense	Deletes a license.
getAdminLicense	Retrieves the admin license, which can be used when all concurrent user licenses are in use.
getCertificates	Returns the certificates in the trust chain for the server.
getLicense	Retrieves information for one license.
getLicenses	Retrieves all license data.
getLicenseUsage	Retrieves the current license usage.
getServerInfo	Retrieves information about server ports and message delivery.
getServerStatus	Returns the status of the server.
getVersions	Retrieves server version information.
importLicenseData	Imports one or more licenses.
logMessage	Enters a message in the server log.
setLogLevel	Changes the log level of a logger.
shutdownServer	Shuts down the ElectricFlow server.
tunePerformance	Adjusts how the server is performing.

Snapshot Management

Commands	Description
createSnapshot	Creates a new snapshot of the specified application.
deleteSnapshot	Deletes snapshot from an application.
getPartialApplicationRevision	Retrieves a partial application when a snapshot is created.
getSnapshot	Find a snapshot by name.
getSnapshotEnvironments	Retrieves a list of environments deployed in the specified snapshot.
getSnapshots	Retrieves all the snapshots in an application.
modifySnapshot	Modifies an existing snapshot of an application.

Tier Map Management

Commands	Description
createTierMap	Creates a tier map for an application.
deleteTierMap	Deletes a tier map from an application.
deleteTierMapping	Deletes a tier mapping from a tier map.
getTierMaps	Retrieves all tier maps that are used by an application.
modifyTierMap	Modifies an existing tier map.

User/Group Management

Commands	Description
login	Logs into the server and saves the session ID for subsequent ectool use. The username provided determines the permissions for commands that can be run during the session.
logout	Logs out of the client session.
addUsersToGroup	Adds ones or more specified users to a particular group.

Commands	Description
createGroup	Creates a new local group of users.
createUser	Creates a new local user.
deleteGroup	Deletes a local group.
deleteUser	Deletes a local user.
getGroup	Retrieves a group by its name.
getGroups	Retrieves all groups.
getUser	Retrieves a user by name.
getUsers	Retrieves all users.
login	Logs into the server and saves the session ID for subsequent ectool use. The username provided determines the permissions for commands that can be run during the session.
logout	Logs out of the client session.
modifyGroup	Modifies an existing group.
modifyUser	Modifies an existing user.
removeUsersFromGroup	Removes one or more users from a particular group.

Workflow Definition Management

Commands	Description
createStateDefinition	Creates a new state definition for a workflow definition.
createTransitionDefinition	Creates a new transition definition for a workflow definition.
createWorkflowDefinition	Creates a new workflow definition for a project.
deleteStateDefinition	Deletes a state definition.
deleteTransitionDefinition	Deletes a transition definition.
deleteWorkflowDefinition	Deletes a workflow definition, including all state and transition definitions.

Commands	Description
getStateDefinition	Finds a state definition by name.
getStateDefinitions	Retrieves all state definitions in a workflow definition.
getTransitionDefinition	Finds a transition definition by name.
getTransitionDefinitions	Retrieves all transition definitions in a workflow definition.
getWorkflowDefinition	Finds a workflow definition by name.
getWorkflowDefinitions	Retrieves all workflow definitions in a project.
modifyStateDefinition	Modifies an existing state definition.
modifyTransitionDefinition	Modifies an existing transition definition.
modifyWorkflowDefinition	Modifies an existing workflow definition.
moveStateDefinition	Moves a state definition within a workflow definition.
moveTransitionDefinition	Moves a transition definition within a workflow definition.

Workflow Management

Commands	Description
completeWorkflow	Marks a workflow as complete, which means transitions are no longer evaluated.
deleteWorkflow	Deletes a workflow, including all states and transitions.
getState	Finds a state by name.
getStates	Retrieves all states in a workflow.
getTransition	Finds a transition by name.
getTransitions	Retrieves all transitions in a workflow.
getWorkflow	Finds a workflow by name.
getWorkflows	Retrieves all workflow instances in a project.
runWorkflow	Runs the specified workflow definition, returns the workflow name.
transitionWorkflow	Manually transition from a workflow active state.

Workspace Management

Commands	Description
`createWorkspace`	Creates a new workspace.
`deleteWorkspace`	Deletes a workspace.
`getWorkspace`	Retrieves a workspace by name.
`getWorkspaces`	Retrieves all workspaces.
`modifyWorkspace`	Modifies an existing workspace.
`resolveFile`	Resolves the path to a log file or artifact in a workspace.

Miscellaneous

Commands	Description
`acquireNamedLock`	Retrieves the named lock.
`changeOwner`	Changes the owner of an object.
`clone`	Makes a copy of an existing ElectricFlow project, procedure, step, schedule, resource, directory provider, email configuration, or email notifier.
`countObjects`	Returns the count of objects specified by the provided filter.
`deleteObjects`	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.
`dumpHeap`	Captures a Java heap dump.
`dumpStatistics`	Prints (emits) internal timing statistics.
`evalDsl`	Evaluates and runs an ElectricFlow domain-specific language (DSL) script.

Commands	Description
`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.
`export`	Exports part or all server data to an XML file. The default is to export all data in the system—the specified path is interpreted by the server. If the path is local, it will be created on the server machine. If it is a network path, it must be accessible by the server and the server user. If it is a relative path (NOT RECOMMENDED), it must be relative to the server's working directory.
`findObjects`	Finds several different types of ElectricFlow objects—it is the underlying mechanism used to implement the ElectricFlow "Search" feature. Because of this command's general nature and the complexity of specifying filter and sort criteria, it is not supported by ectool. Use the Perl API for the `findObjects` command.
`finishCommand`	The agent uses this command to indicate that a command has been run.
`generateDsl`	Generates domain-specific language (DSL) script for an existing object.
`getObjects`	Retrieves the full object based on IDs returned by `findObjects`. All requested objects must be of the same `objectType`. See `findObjects` for a list of object types.
`graphStateMachine`	Generates a graph element with a stateMachine DOT graph as CDATA content.
`import`	Imports data from an XML export file.
`logStatistic`	Prints (emits) a statistics value to StatsD.
`releaseNamedLock`	Releases the named lock that synchronizes the name of an object.

API Commands - ACL Management

deleteAclEntry on page 71

getAccess on page 76

getAclEntry on page 81

modifyAclEntry on page 86

restoreAclInheritance on page 92

breakAclInheritance

Breaks access control list (ACL) inheritance at the specified object. When inheritance is 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
`Locator arguments:`	
`applicationName`	The name of the application. Argument type: String
`applicationTierName`	The name of the application tier. Argument type: String
`artifactName`	The name of the artifact. Argument type: String
`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 ElectricFlow server interprets the name form correctly. Argument type: String
`componentName`	The name of the component. Argument type: String
`configName`	The name of the email configuration. Argument type: String

Arguments	Descriptions
`credentialName`	The name of the credential that can be one of these formats: • **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 project where the target object is. Argument type: String
`environmentName`	The name of the environment. Argument type: String
`environmentTemplateName`	The name of the environment template. Argument type: String
`environmentTemplateTierName`	The name of the environment template tier. Argument type: String
`environmentTierName`	The name of the environment tier. Argument type: String
`flowName`	The name of the flow. Argument type: String
`flowRuntimeName`	The name of the flow runtime. Argument type: String
`flowStateName`	The name of the flow state. Argument type: String
`flowTransitionName`	The name of the flow transition. Argument type: String
`gatewayName`	The name of the gateway. Argument type: String
`groupName`	The full name of the group. For Active Directory and LDAP, the full name if the full domain name. Argument type: String

Arguments	Descriptions
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notifierName`	The name of the email notifier. Argument type: String
`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. Argument type: String
`objectType`	(Optional) The type of object. Argument type: String
`path`	The property path. Argument type: String
`pipelineName`	The name of the pipeline. Argument type: String
`pluginName`	The plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin. Argument type: String
`procedureName`	The name of the procedure or a path to a procedure that includes the name. If you use this argument, you must also use `projectName`. Argument type: String
`processName`	The name of the process if the container is a process or process step. Argument type: String
`processStepName`	The name of the process step if the container is a process step. Argument type: String

Arguments	Descriptions
`projectName`	The name of the project, which can be a path. The project name is ignored for credentials, procedures, steps, and schedules when they are specified as a path. Argument type: String
`propertySheetId`	The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`releaseName`	The name of the Release. Argument type: String
`repositoryName`	The name of the repository used for artifact management. Argument type: String
`resourceName`	The name of the resource. Argument type: String
`resourcePoolName`	The name of the resource pool. Argument type: String
`resourceTemplateName`	The name of the resource template.
`scheduleName`	The name of a schedule, which can be a path to the schedule. If you use this argument, you must also use `projectName`. Argument type: String
`snapshotName`	The name of the snapshot. Argument type: String
`stageName`	The name of the stage in a pipeline. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`stateName`	The name of the state. Argument type: String

Arguments	Descriptions
stepName	The name of the step, which can be a path to the step. If you use this argument, you must also use `projectName` and `procedureName`. Argument type: String
systemObjectName	The name of the system object. System objects names include: `admin|artifactVersions|directory|emailConfigs|log|plugins|server|session|workspaces`. Argument type: SystemObjectName
taskName	The name of the task in a pipeline stage. Argument type: String
transitionDefinitionName	The name of the transition definition. Argument type: String
transitionName	The name of the transition. Argument type: String
userName	The full name of a user. For Active Directory or LDAP, this may be `user@domain`). Argument type: String
workflowDefinitionName	The name of the workflow definition. Argument type: String
workflowName	The name of the workflow. Argument type: String
workspaceName	The name of a workspace. Argument type: String
zoneName	The name of the zone. Argument type: String

Positional arguments

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

Response

None or status OK message.

ec-perl

syntax: `$<object>->breakAclInheritance({<optionals>});`

Example

```
$cmdr->breakAclInheritance ({projectName => "Default", pipelineName => "Q1 2-16 Tra
ding System"});
```

ectool

syntax: `ectool breakAclInheritance [optionals]`

Example

```
ectool breakAclInheritance --projectName "Default" --pipeline "Q1 2-16 Trading Syst
em"
```

Back to Top

checkAccess

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

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

Arguments	**Descriptions**
`Locator arguments:`	
`applicationName`	The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	The name of the application tier. Argument type: String
`artifactName`	The name of the artifact. Argument type: String
`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 when you specify its name one of these ways. The ElectricFlow server interprets the name form correctly. Argument type: String
`componentName`	The name of the component. Argument type: String

Arguments	Descriptions
`configName`	The name of the email configuration. Argument type: String
`credentialName`	The name of the credential container of the property sheet that owns the property. Specify `credentialName` using one of two forms: • **relative** (for example, *"cred1"*)–The credential is assumed to be in the project that contains the request target object. This form requires a qualifying project name. • **absolute** (for example, *"/projects/BuildProject/credentials/cred1"*)–The credential can be from any specified project, regardless of the project target object project. Argument type: String
`environmentName`	The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	The name of the environment template. Argument type: String
`environmentTemplateTierName`	The name of the environment template tier. Argument type: String
`environmentTierName`	The name of the environment tier. Argument type: String
`flowName`	The name of the flow that must be unique within the project. Argument Type: String
`flowRuntimeName`	The name of the flow runtime that must be unique within the flow. Argument Type: String
`flowRuntimeStateName`	The name of the flow run-time state. Argument Type: String
`flowStateName`	Name of the flow state that must be unique within the flow. Argument Type: String
`flowTransitionName`	Name of the flow transition that must be unique within the flow state. Argument Type: String

Arguments	Descriptions
`gatewayName`	The name of the gateway. Argument type: String
`groupName`	The full name of the group. For Active Directory and LDAP, this is a full domain name. Argument type: String
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notifierName`	The name of the email notifier. Argument type: String
`objectId`	The 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. Argument type: String
`objectType`	The type of object. Argument type: String
`path`	Property path string. Argument type: String
`pipelineName`	The name of the pipeline. Argument type: String
`pluginName`	The name of the plugin. This is the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin. Argument type: String
`procedureName`	The name of the procedure. It can be be a path to the procedure. When using this procedure, you must also use `projectName`. Argument type: String

Arguments	Descriptions
`processName`	The name of the process. Argument type: String
`processStepName`	The name of the process step. Argument type: String
`projectName`	The name of the project that must be unqiue among all projects. It can be a path to the project. The project name is ignored for credentials, procedure, steps, and schedules when it is specified as a path. Argument type: String
`propertySheetId`	The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`releaseName`	The name of the Release that owns the property. Argument type: String
`repositoryName`	The name of the repository for artifact management. Argument type: String
`resourceName`	The name of the resource. Argument type: String
`resourcePoolName`	The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	The name of the resource template. Argument type: String
`scheduleName`	The name of the schedule, which can be the path to the schedule. When using this argument, you must also enter `projectName`. Argument type: String
`snapshotName`	The name of the snapshot, which can be the path to the snapshot. Argument type: String
`stageName`	The name of the stage. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String

Arguments	Descriptions
`stateName`	The name of the state. Argument type: String
`stepName`	The name of the step. It can be a path to the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`systemObjectName`	System object names include: `admin|directory|licensing|log|plugins|priority|projects|`. Argument type: SystemObjectName
`taskName`	The name of the task in a stage in a pipeline. Argument type: String
`transitionDefinitionName`	The name of the transition definition. Argument type: String
`transitionName`	The name of the transition. Argument type: String
`userName`	The full name of the user. For Active Directory and LDAP, the name can be `user@domain`. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`workflowName`	The name of the workflow. Argument type: String
`workspaceName`	The name of the workspace. Argument type: String
`zoneName`	The name of the zone. Argument type: String

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: `$<object>->checkAccess ({<optionals>});`

Example

```
$cmdr->checkAccess ({projectName =>"Production"});
```

ectool

syntax: `ectool checkAccess [optionals]`

Example

```
ectool checkAccess --projectName "Production"
```

Back to Top

createAclEntry

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

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

Arguments	Descriptions
`principalType`	This is either `user` or `group`. Argument type: PrincipalType
`principalName`	This is either a user or a group name. Argument type: PrincipalName
`Locators:`	
`applicationName`	The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	The name of the application tier. Argument type: String
`artifactName`	The name of the artifact. Argument type: String

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 ElectricFlow server interprets the name form correctly. Argument type: String
`changePermissionsPrivilege`	<`allow|deny`> –Determines whether the principal can modify access control for the object. Argument type: Access
`componentName`	The name of the component. Argument type: String
`configName`	The name of the email configuration (`emailConfig`). Argument type: String
`credentialName`	The name of the credential specified in one of these formats: • **relative**(for example, *"cred1"*)–The credential is assumed to be in the project that contains the requested target object. • **absolute**(for example, *"/projects/BuildProject/credentials/cred1"*)–The credential can be from any specified project, regardless of the project for the target object. When using this argument, you must also enter `projectName`. Argument type: String
`environmentName`	The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	The name of the environment template. Argument type: String
`environmentTemplateTierName`	The name of the environment template tier. Argument type: String
`environmentTierName`	The name of the environment tier. Argument type: String

Arguments	Descriptions
`executePrivilege`	<`allow|deny`> –Determines whether the principal can invoke this object as part of a job. This privilege is only relevant for a few objects such as procedures and procedure steps. Argument type: Access
`flowName`	The name of the flow. Argument: String
`flowRuntimeName`	The name of the flow runtime. Argument: String
`flowRuntimeStateName`	The name of the flow state. Argument: String
`flowStateName`	The name of the flow state. Argument: String
`flowTransitionName`	The name of the flow transition. Argument: String
`gatewayName`	The name of the gateway. Argument type: String
`groupName`	The name of a group. Argument type: String
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`modifyPrivilege`	<`allow|deny`> –Determines whether the principal can change the contents of the object. Argument type: Access
`notifierName`	The name of the email notifier. Argument type: String

Arguments	Descriptions
`objectId`	The object identifier returned by `findObjects` and `getObjects`. Argument type: String
`objectType`	(Optional) The type of object. Argument type: String
`path`	The path to the property. Argument type: String
`pipelineName`	The name of the pipeline. Argument type: String
`pluginName`	The name of the plugin. It is the plugin key for a promoted plugin or the plugin key and version for an unpromoted plugin. Argument type: String
`procedureName`	The name of the procedure. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	The name of the process. Argument type: String
`processStepName`	The name of the process step. Argument type: String
`projectName`	The name of the project. Argument type: String
`propertySheetId`	The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`readPrivilege`	<`allow|deny`> –Determines whether the principal can examine the contents of the object. Argument type: Access
`releaseName`	The name of the Release. Argument type: String
`repositoryName`	The name of the repository for artifact management. Argument type: String

Arguments	Descriptions
`resourceName`	The name of the resource. Argument type: String
`resourcePoolName`	The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	The name of the resource template. Argument type: String
`scheduleName`	The name of the schedule. When using this argument, you must also enter `projectName`. Argument type: String
`snapshotName`	The name of the snapshot. Argument type: String
`stageName`	The name of the stage. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`stateName`	The name of the state. Argument type: String
`stepName`	The name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`systemObjectName`	System object names include: `admin|artifacts|directory|emailConfigs|forceAbort|licensing|log|plugins|priority|projects|repositories|resources|server| session|test|workspaces|zonesAndGateways` Argument type: SystemObjectName
`taskName`	The name of the task. Argument type: String
`transitionDefinitionName`	The name of the transition definition. Argument type: String

Arguments	Descriptions
`transitionName`	The name of the transition. Argument type: String
`userName`	The full name of the user. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`workflowName`	The name of the workflow. Argument type: String
`workspaceName`	The name of the workspace. Argument type: String
`zoneName`	The name of the zone. Argument type: String

Positional arguments

`principalType`, `principalName`, and locator options.

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool createAclEntry <principalType> <principalName> [optionals]`

Example

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

Back to Top

deleteAclEntry

Deletes an access control entry (ACE) in an access control list (ACL) on an object for a given principal (user or group).

You must specify `principalType`, `principalName`, and locator arguments.

Arguments	Descriptions
`principalType`	A user or a group: `<user|group>`. The default is to `user`. Argument type: PrincipalType
`principalName`	The name of the user or the group. Argument type: PrincipalName
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets the name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String

Arguments	Descriptions
`credentialName`	(Optional) The name of the credential specified in one of these formats: • **relative**(for example, *"cred1"*)–The credential is assumed to be in the project that contains the requested target object. • **absolute**(for example, *"/projects/BuildProject/credentials/cred1"*)–The credential can be from any specified project, regardless of the project for the target object. Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) The name of the environment template that must be unique among all projects. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String

Arguments	Descriptions
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of a group whose ACL entry you want to delete. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notifierName`	(Optional) The name of the email notifier with the ACE that you want to delete. Argument type: String
`objectId`	(Optional) An object identifier returned by `findObjects` and `getObjects`. Argument type: String
`objectType`	(Optional) The type of object. Argument type: String
`path`	(Optional) Path to the property. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin with the ACE that you want to delete. Argument type: String
`procedureName`	(Optional) The name of the procedure with the ACE that you want to delete. When you use this argument, you must also enter `projectName` for the project of which this procedure is a member. Argument type: String

Arguments	Descriptions
`processName`	(Optional) The name of the process. Argument type: String
`processStepName`	(Optional) The name of the process step. Argument type: String
`projectName`	(Optional) The name of the project with the ACE that you want to delete. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`releaseName`	(Optional) The name of the Release which owns the property. Argument type: String
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of the resource with the ACE that you want to delete. Argument type: String
`resourcePoolName`	(Optional) The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument Type: String
`scheduleName`	(Optional) The name of the schedule with the ACE that you want to delete. When you use this argument, you must also enter the `projectName` from which this schedule runs procedures. Argument type: String
`snapshotName`	(Optional) The name of the snapshot. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String

Arguments	Descriptions
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step with the ACE that you want to delete. When using this argument, you must also enter `projectName` and `procedureName` to indicate where this step resides. Argument type: String
`systemObjectName`	(Optional) System object names include: `admin|directory|licensing|log|plugins|priority|projects|resources|server|session|workspaces` Argument type: SystemObjectName
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The name of the user with the ACE that you want to delete. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String
`workspaceName`	(Optional) The name of the workspace with the ACL entry that you want to delete. Argument type: String
`zoneName`	(Optional) The name of the zone. Argument type: String

Positional arguments

```
principalType, principalName
```

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteAclEntry <principalType> <principalName> [optionals]`

Example

```
ectool deleteAclEntry "user" "j smith" --projectName "Default"
```

Back to Top

getAccess

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

You must specify object locators to find the object to which you need to verify access.

Arguments	Descriptions
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets the name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String

Arguments	Descriptions
`configName`	(Optional) The name of the email configuration. Argument type: String
`credentialName`	(Optional) The name of the credential specified in one of these formats: • **relative**(for example, *"cred1"*)–The credential is assumed to be in the project that contains the requested target object. • **absolute**(for example, *"/projects/BuildProject/credentials/cred1"*)–The credential can be from any specified project, regardless of the project for the target object. Argument type: String
`emulateRestoreInheritance`	(Optional) Whether or not to include one level of broken inheritance if it exists. This argument is 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 `true` or `1`, this argument returns ACL information to what it would be if inheritance were restored on this object. Argument type: Boolean
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String

Arguments	Descriptions
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of the group. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: String
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String
`notifierName`	(Optional) The name of the email notifier with the ACL. Argument type: String
`objectId`	(Optional) An object identifier returned by `findObjects` and `getObjects`. Argument type: String
`objectType`	(Optional) The type of object. Argument type: String
`path`	(Optional) Property path. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin with the ACL. Argument type: String

Arguments	Descriptions
`procedureName`	(Optional) The name of the procedure with the ACL. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) The name of the process. Argument type: String
`processStepName`	(Optional) The name of the process step. Argument type: String
`projectName`	(Optional) The name of the project that contains the ACL that must be unique among all projects. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`releaseName`	(Optional) The name of the Release. Argument type: String
`repositoryName`	(Optional)The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional)The name of the resource with the ACL. Argument type: String
`resourcePoolName`	(Optional)The name of a pool with one or more resources. Argument type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of the schedule with the ACL. **Also requires** `projectName`
`snapshotName`	(Optional) The name of a snapshot. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition.

Arguments	Descriptions
stateName	(Optional) The name of the state.
stepName	(Optional) The name of the step containing the ACL. When using this argument, you must also enter projectName Argument type: String
systemObjectName	(Optional) System objects include: admin|artifactVersions|directory|emailConfigs|log|p lugins| server|session|workspaces Argument type: SystemObjectName
taskName	(Optional) The name of the task. Argument type: String
transitionDefinitionName	(Optional) The name of the transition definition. Argument type: String
transitionName	(Optional) The name of the transition. Argument type: String
userName	(Optional) The name of the user with the ACL. Argument type: String
workflowDefinitionName	(Optional) The name of the workflow definition. Argument type: String
workflowName	(Optional) The name of the workflow. Argument type: String
workspaceName	(Optional) The name of the workspace with the ACL. Argument type: String
zoneName	(Optional) The name of the zone. Argument type: String

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

Example

```
$cmdr->getAccess({projectName => "Quarterly Summary Results"});
```

ectool

syntax: `ectool getAccess [optionals]`

Example

```
ectool getAccess --projectName "Quarterly Summary Results"
```

Back to Top

getAclEntry

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

You must specify a `principalType`, `principalName`, and an object locator to specify the ACE.

Arguments	Descriptions
`principalType`	Type of principal for this ACE: `user` or `group`. Argument type: PrincipalType
`principalName`	Name of the user or group for the ACE. Argument type: PrincipalName
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String

Arguments	Descriptions
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets the name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String
`credentialName`	(Optional) The name of the credential specified in one of these formats: • **relative**(for example, *"cred1"*)–The credential is assumed to be in the project that contains the requested target object. • **absolute**(for example, *"/projects/BuildProject/credentials/cred1"*)–The credential can be from any specified project, regardless of the project for the target object. Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) Name of the environment tier. Argument type: String
`flowName`	Name of the flow that must be unique within the project. Argument Type: String

Arguments	Descriptions
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	Name of the flow state that must be unique within the flow. Argument Type: String
`flowTransitionName`	Name of the flow transition that must be unique within the flow state. Argument Type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of the group. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String
`notifierName`	(Optional) The name of the email notifier. Argument type: String
`objectId`	(Optional) This is an object identifier returned by `findObjects` and `getObjects`. Argument type: String
`objectType`	(Optional) The type of object. Argument type: String
`path`	(Optional) The property path. Argument type: String

Arguments	Descriptions
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin. The plugin key for a promoted plugin or the plugin key and version for an unpromoted plugin. Argument type: String
`procedureName`	(Optional) The name of the procedure with the ACL. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) The name of the process. Argument type: String
`processStepName`	(Optional) The name of the process step. Argument type: String
`projectName`	(Optional) The name of the project. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: String
`releaseName`	(Optional) The name of the Release. Argument type: String
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: UUID
`resourceName`	(Optional) The name of the resource. Argument type: String
`resourcePoolName`	(Optional) The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	(Optional) The name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of a schedule. When using this argument, you must also enter `projectName`. Argument type: String

Arguments	Descriptions
`snapshotName`	(Optional) The name of a snapshot. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String

`systemObjectName`	(Optional) System objects include:
	`admin|artifactVersions|directory|emailConfigs|log|plugins|`
	`server|session|workspaces`
	
	Argument type: SystemObjectName

`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The full name of the user. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String
`workspaceName`	(Optional) The name of the workspace. Argument type: String

Arguments	Descriptions
`zoneName`	(Optional) The name of the zone. Argument type: String

Positional arguments

`principalType`, `principalName`, and arguments to specify the object, beginning with the top-level object locator.

Response

One `aclEntry` element.

ec-perl

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

Example

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

ectool

syntax: `ectool getAclEntry <principalType> <principalName> [optionals]`

Example

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

Back to Top

modifyAclEntry

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

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

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

Arguments	Descriptions
`principalType`	Type of principal for this access control entry: `user` or `group`. Argument type: PrincipalType
`principalName`	Name of the user or group for this access control entry. Argument type: PrincipalName
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String

Arguments	Descriptions
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets the name form correctly. Argument type: String
`changePermissionsPrivilege`	(Optional) <`allow|deny`> –Determines whether the principal can modify access control for the object. Argument type: Access
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String
`credentialName`	(Optional) The name of the credential specified in one of these formats: • **relative**(for example, *"cred1"*)–The credential is assumed to be in the project that contains the requested target object. • **absolute**(for example, *"/projects/BuildProject/credentials/cred1"*)–The credential can be from any specified project, regardless of the project for the target object. Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) The name of the environment template. Argument type: String

Arguments	Descriptions
`environmentTemplateTierName`	(Optional) The name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`executePrivilege`	(Optional) <`allow|deny`> –Determines whether the principal can invoke this object as part of a job. This privilege is only relevant for a few objects such as procedures and procedure steps. Argument type: Access
`flowName`	(Optional) The name of the flow that must be unique within the project. Argument Type: String
`flowRuntimeName`	(Optional) The name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) The name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state that must be unique within the flow. Argument Type: String
`flowTransitionName`	(Optional) The name of the flow transition that must be unique within the flow state. Argument Type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of the group with the ACE. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID

Arguments	Descriptions
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`modifyPrivilege`	(Optional) <`allow|deny`> –Determines whether the principal can change the contents of the object. Argument type: Access
`notifierName`	(Optional) The name of the email notifier with the ACE. Argument type: String
`objectId`	(Optional) The object identifier returned by `findObjects` and `getObjects`. Argument type: String
`objectType`	(Optional) The type of object. Argument type: String
`path`	(Optional) The property path. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin with the ACE. Argument type: String
`procedureName`	(Optional) The name of the procedure with the ACL entry. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) The name of the process. Argument type: String
`processStepName`	(Optional) The name of the process step. Argument type: String
`projectName`	(Optional) The name of the project with the ACE. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID

Arguments	Descriptions
readPrivilege	(Optional) <allow|deny> –Determines whether the principal can examine the contents of the object. Argument type: Access
releaseName	(Optional) The name of the Release. Argument type: String
repositoryName	(Optional) The name of the repository for artifact management. Argument type: String
resourceName	(Optional) The name of the resource containing the ACE. Argument type: String
resourcePoolName	(Optional) The name of a resource pool. Argument type: String
resourceTemplateName	(Optional) The name of the resource template. Argument type: String
scheduleName	(Optional) The name of the schedule with the ACE. When using this argument, you must also enter projectName. Argument type: String
snapshotName	(Optional) The name of a snapshot. Argument type: String
stageName	(Optional) The name of the stage. Argument type: String
stateDefinitionName	(Optional) The name of the state definition. Argument type: String
stateName	(Optional) The name of the state. Argument type: String
stepName	(Optional) The name of the step with the ACE. When using this argument, you must also enter projectName. Argument type: String

Arguments	Descriptions
systemObjectName	(Optional) System object names include: `admin|artifacts|directory|emailConfigs|forceAbort|licensing|log|plugins|priority|projects|repositories|resources|server|session|test|workspaces|zonesAndGateways` Argument type: SystemObjectName
taskName	(Optional) The name of the task in a stage in a pipeline. Argument type: String
transitionDefinitionName	(Optional) The name of the transition definition. Argument type: String
transitionName	(Optional) The name of the transition. Argument type: String
userName	(Optional) The username containing the ACE. Argument type: String
workflowDefinitionName	The name of the workflow definition. Argument type: String
workflowName	The name of the workflow. Argument type: String
workspaceName	The name of the workspace containing the ACE. Argument type: String
zoneName	The name of the zone. Argument type: String

Positional arguments

`principalType, principalName`, and arguments to specify the object, beginning with the top-level object locator

Response

Retrieves a modified ACE element..

ec-perl

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

Example

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

ectool

syntax: `ectool modifyAclEntry <principalType> <principalName> [optionals]`

Example

```
ectool modifyAclEntry "user" "j smith" --projectName "Sample Project"
   --snapshotName "LastGood"
```

Back to Top

restoreAclInheritance

Restores the 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
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets the name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String

Arguments	Descriptions
`credentialName`	(Optional) The name of the credential specified in one of these formats: • **relative**(for example, *"cred1"*)–The credential is assumed to be in the project that contains the requested target object. • **absolute**(for example, *"/projects/BuildProject/credentials/cred1"*)–The credential can be from any specified project, regardless of the project for the target object. When using this argument, you must also enter `projectName`. Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`flowName`	(Optional) Name of the flow that must be unique within the project. Argument Type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) Name of the flow state that must be unique within the flow. Argument Type: String
`flowTransitionName`	Name of the flow transition that must be unique within the flow state. Argument Type: String

Arguments	Descriptions
gatewayName	(Optional) The name of the gateway. Argument type: String
groupName	(Optional) The name of the group with the ACL inheritance that you want to restore. Argument type: String
jobId	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
jobStepId	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
notifierName	(Optional) The name of the email notifier with the ACL inheritance that you want to restore. **Also requires** projectName and procedureName; projectName, procedureName, and stepName; jobId or jobStepId Argument type: String
objectId	(Optional) This is an object identifier returned by findObjects and getObjects. Argument type: String
objectType	(Optional) The type of object. Argument type: String
path	(Optional) Property path. Argument type: String
pipelineName	(Optional) The name of the pipeline. Argument type: String
pluginName	(Optional) The name of the plugin with the ACL inheritance that you want to restore. Argument type: String

Arguments	Descriptions
`procedureName`	(Optional) The name of the procedure with the ACL inheritance that you want to restore. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) The name of the process. Argument type: String
`processStepName`	(Optional) The name of the process step. Argument type: String
`projectName`	(Optional) The name of the project with the ACL inheritance that you want to restore. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`releaseName`	(Optional) The name of the Release. Argument type: String
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of the resource whose ACL inheritance you want to restore. Argument type: String
`resourcePoolName`	(Optional) The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of the schedule with the ACL inheritance that you want to restore. When using this argument, you must also enter `projectName`. Argument type: String
`snapshotName`	(Optional) The name of a snapshot. Argument type: String

Arguments	Descriptions
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step with the ACL inheritance that you want to restore. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`systemObjectName`	(Optional) 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` Argument type: SystemObjectName
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The name of the user with the ACL inheritance that you want to restore. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String
`workspaceName`	(Optional) The name of the workspace with the ACL inheritance that you want to restore. Argument type: String

Arguments	Descriptions
`zoneName`	(Optional) The name of the zone. Argument type: String

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 => "Software tools"});
```

ectool

syntax: `ectool restoreAclInheritance [optionals]`

Example

```
ectool restoreAclInheritance --projectName "Software tools"
```

Back to Top

API Commands - Applications

createApplication

Creates a new application for a project.

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

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

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

Positional arguments

`projectName`, `applicationName`

Response

Returns an application element.

ec-perl

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

Example

```
$ec->createApplication("Default", "Deploy", {description => "QA testing"});
```

ectool

syntax: `ectool createApplication <projectName> <applicationName> [optionals]`

Example

```
ectool createApplication "Default" "Deploy" --description "QA testing"
```

Back to Top

deleteApplication

Deletes an application.

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

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

Positional arguments

`projectName`, `applicationName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

```
ectool deleteApplication "Default" "Undeploy"
```

Back to Top

getApplication

Retrieves an application by name.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application that must be unique among all projects. Argument Type: String
`applicationEntityRevisionId`	(Optional) The revision ID of the versioned object. Argument type: UUID

Positional arguments

`projectName`, `applicationName`

Response

Retrieves the specified application element.

ec-perl

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

Example

```
$ec->getApplication("Default", "Deploy", {applicationEntityRevisionId => "4fa765dd-
73f1-11e3-b67e-b0a420524153"});
```

ectool

syntax: `ectool getApplication <projectName> <applicationName> [optionals]`

Example

```
ectool getApplication "Default" "Deploy" --applicationEntityRevisionId 4fa765dd-73f
1-11e3-b67e-b0a420524153
```

Back to Top

getApplications

Retrieves all applications in a project.

You must specify the `projectName`.

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`includeEntityRevisions`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, the search results include the revisions of application. Argument type: Boolean
`referenceComponentName`	(Optional) Name of the master component. Argument Type: String
`referenceComponentProject`	(Optional) Project to which the master component belongs. Argument Type: String

Positional arguments

`projectName`

Response

Retrieves zero or more application elements.

ec-perl

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

Example

```
$ec->getApplications("Default", {referenceComponentProject => "Financial Summarie
s"});
```

ectool

syntax: `ectool getApplications <projectName> [optionals]`

Example

```
ectool getApplications "Default" --referenceComponentProject "Financial Summaries"
```

Back to Top

modifyApplication

Modifies an existing application.

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

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

Positional arguments

`projectName`, `applicationName`

Response

Returns a modified application element.

ec-perl

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

Example

```
$ec->modifyApplication("Default", "Deploy config", {newName=> "Deploy WAR file", de
scription => "Pet store website"});
```

ectool

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

Example

```
ectool modifyApplication "Default" "Deploy config" --newName "Deploy WAR file" --de
scription "Pet Store website"
```

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	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application that must be unique among all projects. Argument Type: String
`applicationTierName`	Name of the tier that must be unique within the application. Argument Type: String
`description`	(Optional) Comment text describing this object that is not interpreted at all by ElectricFlow. Argument Type: String

Positional arguments

`projectName` , `applicationName`, `applicationTierName`

Response

Returns an application tier element.

ec-perl

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

Example

```
$ec->createApplicationTier("Default", "Deploy", "Heat Clinic", {description => "Web
setup"});
```

ectool

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

Example

```
ectool createApplicationTier "Default" "Deploy" "Heat Clinic" --description "Web se
tup"
```

Back to Top

deleteApplicationTier

Deletes a tier from an application.

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

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

Positional arguments

`projectName` , `applicationName`, `applicationTierName`

Response

None or a status OK message.

ec-perl

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

Example

```
$ec->deleteApplicationTier("Default", "Undeploy", "Tomcat");
```

ectool

syntax: `ectool deleteApplicationTier <projectName> <applicationName> <applicationTierName>`

Example

```
ectool deleteApplicationTier "Default" "Undeploy" "Tomcat"
```

Back to Top

getApplicationTier

Retrieves an application tier by name.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application that must be unique among all projects. Argument Type: String
`applicationTierName`	Name of the tier that must be unique within the application. Argument Type: String
`applicationEntityRevisionId`	(Optional) The revision ID of the versioned object. Argument type: UUID

Positional arguments

`projectName` , `applicationName`, `applicationTierName`

Response

Returns an application tier element.

ec-perl

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

Example

```
$ec->getApplicationTier("Default", "Deploy", "Tomcat", {applicationEntityRevisionId
=> "4fa765dd-73f1-11e3-b67e-b0a420524153"});
```

ectool

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

Example

```
ectool getApplicationTier "Default" "Deploy" "Tomcat" --applicationEntityRevisionId
4fa765dd-73f1-11e3-b67e-b0a420524153
```

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 that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application that must be unique among all projects. Argument Type: String
`applicationEntityRevisionId`	(Optional) The revision ID of the versioned object. Argument type: UUID

Positional arguments

`projectName, applicationName`

Response

Returns zero or more application tier elements.

ec-perl

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

Example

```
$ec->getApplicationTiers("Default", "Deploy", {applicationEntityRevisionId => "4fa7
65dd-73f1-11e3-b67e-b0a420524153"});
```

ectool

syntax:`ectool getApplicationTiers <projectName> <applicationName> [optionals]`

Example

```
ectool getApplicationTiers "Default" "Deploy" --applicationEntityRevisionId 4fa765d
d-73f1-11e3-b67e-b0a420524153
```

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 that must be unique among all projects. Argument Type: String

Arguments	Descriptions
`componentName`	Name of the component. Argument Type: String
`applicationEntityRevisionId`	(Optional) The revision ID of the versioned object. Argument type: UUID
`applicationName`	(Optional) Name of an application to which this component is scoped. Argument Type: String

Positional arguments

`projectName, componentName`

Response

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

ec-perl

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

Example

```
$ec->getApplicationTiersInComponent("Default", "WAR file", {applicationName => "Sna
pshot"});
```

ectool

syntax:`ectool getApplicationTiersInComponent <projectName> <componentName> [optionals]`

Example

```
ectool getApplicationTiersInComponent "Default" "WAR file" --applicationName "Snaps
hot"
```

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 that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application that must be unique within the project. Argument Type: String
`applicationTierName`	Name of the tier that must be unique within the application. Argument Type: String
`description`	(Optional) Comment text describing this object, which is not interpreted by ElectricFlow. Argument Type: String
`newName`	(Optional) New name for an existing object that is being renamed. Argument Type: String

Positional arguments

`projectName, applicationName, applicationTierName`

Response

Returns an updated application tier element.

ec-perl

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

Example

```
$ec->modifyApplicationTier("Default", "Publish README", "App Server", {newName=> "D
eploy README", description=> "Revised README file"});
```

ectool

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

Example

```
ectool  modifyApplicationTier "Default" "Publish README" "App Server" --newName "De
ploy README" --description "Revised README file"
```

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. 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 ElectricFlow server interprets the name form correctly. Argument type: String
`dependentArtifactVersions`	(Optional) 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. Argument type: Collection

Positional arguments

`artifactVersionName`

Response

Returns one or more dependent artifact versions.

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> [optionals]`

Example

```
ectool addDependentsToArtifactVersion "myGroup:myAKey:1.0.0-55"
   --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 ElectricFlow 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. Argument type: String
`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". Argument type: Boolean

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 ElectricFlow 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. Argument type: String
`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 backing store properly. Argument type:Boolean

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 `groupId` and `artifactKey`.

Arguments	Descriptions
`groupId`	A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, spaces, underscores, hyphens, and periods. Argument type: String
`artifactKey`	A user-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String
`artifactVersionNameTemplate`	(Optional) 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 (Go to **Administration** > **Server**, and select the Settings link). Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String

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> [optionals]`

Example

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

Back to Top

createArtifactVersion

Creates a new artifact version.

You must specify `version`.

Arguments	Descriptions
`version`	The version component of the GAV (`groupId`/`artifactVersionId`/`version`) coordinates. Argument type: String
`artifactKey`	(Optional) A user-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`dependentArtifactVersions`	(Optional) The set of artifact versions on which this `artifactVersion` depends. Argument type: Collection
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`groupId`	(Optional) The `groupId` component of the GAV (`groupId`/`artifactVersionId`/`version`) coordinates. This field is limited to alphanumeric characters, spaces, spaces, underscores, hyphens, and periods. Argument type: String
`jobStepId`	(Optional) The unique identifier for the job step that is used to make a project association. Argument type: UUID
`repositoryName`	(Optional) The name of the artifact repository. Argument type: String

Positional arguments

`version`

Response

Returns an artifactVersion on page 774 element.

ec-perl

syntax:`$cmdr->createArtifactVersion(<version>, {<optionals>});`

Examples

```
$cmdr->createArtifactVersion(1.1, {artifactName => test:test1});
```

```
$cmdr->createArtifactVersion(1.2, {artifactName => test1:test2});
```

ectool

syntax: `ectool createArtifactVersion <version> [optionals]`

Examples

```
ectool createArtifactVersion 1.1 --artifactName test:test1
```

```
ectool createArtifactVersion 1.2 --artifactName test1:test2
```

Back to Top

createRepository

Creates a repository for one or more artifacts.

You must specify a `repositoryName`.

Arguments	Descriptions
`repositoryName`	The name of the artifact repository. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`repositoryDisabled`	(Optional) <*Boolean flag* -`0|1|true|false`> – When the value is `1` or `true`, the repository is disabled. The default is `0` or `false`. Argument type: Boolean
`url`	The URL to use to communicate with the repository server. Argument type: String
`zoneName`	The name of the zone where this repository resides. Argument type: String

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> [optionals]`

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. Argument type: String

Positional arguments

`artifactName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteArtifact <artifactName>`

Example

```
ectool deleteArtifact "ElectricFlow:SDK"
```

Back to Top

deleteArtifactVersion

Deletes artifact version metadata from the ElectricFlow 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 ElectricFlow server interprets the name form correctly. Argument type: String

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

You must enter a `repositoryName`.

Arguments	Descriptions
`repositoryName`	The name of the artifact repository. Argument type: String

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 ElectricFlow 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
`artifactKey`	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. This is the `artifactKey` component of the GAV (`GroupId`/`ArtifactVersionId`/`Version`) coordinates. Argument type: String
`artifactName`	The name of an artifact. Argument type: String
`artifactVersionName`	The name of an artifact version (~~artifactVersion~~). Argument type: String

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 ElectricFlow 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 ElectricFlow 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)` Argument type: Collection
`groupId`	A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods. This is the `groupId` component of the GAV (`GroupId`/`ArtifactVersionId`/`Version`) coordinates. The default is the name of the project that owns the job step requesting the artifact when a job step makes such as request. This argument is required when there is no job step. Argument type: String
`includeDependents`	<*Boolean flag* -`0|1|true|false`> Options are: • `0/false` – dependent artifacts are not retrieved. • `1/true` – dependent artifacts are retrieved. Argument type: Boolean
`jobStepId`	The unique identifier for the job step that is making the request. This job step is marked as a retriever for the matching artifact versions. Argument type: UUID
`versionRange`	The range of versions to search. Version range syntax is standard number interval notation. `()` marks exclusive ranges and `[]` marks inclusive ranges. Argument type: String

Positional arguments

None

Response

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 dependents.

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. Argument type: String

Positional arguments

`artifactName`

Response

Returns 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.

Arguments	Descriptions
None	

Positional arguments

None

Response

Returns 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 ElectricFlow server interprets either name form correctly. Argument type: String
`includeRetrieverJobs`	(Optional) <*Boolean flag -* `0|1|true|false`> If set to `1` or `true`, this argument includes `jobId` and `jobName` in returned information. A retriever job is any job that has retrieved the artifact version. Argument type: Boolean
`includeRetrieverJobSteps`	(Optional) <*Boolean flag -* `0|1|true|false`> If set to `1` or `true`, 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. Argument type: Boolean
`maxRetrievers`	(Optional) If one of the `includeRetriever`* options are specified, return at most "this many" of the most recent retrievers. Without this option, the ElectricFlow server will return all retrievers. Argument type: String

Positional arguments

`artifactVersionName`

Response

Returns 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> [optionals]`

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. Argument type: String
`retrieverJobId`	The job ID to use when retrieving an artifact. Argument type: UUID
`retrieverJobStepId`	The job step ID to use when retrieving an artifact. Argument type: UUID

Positional arguments

None

Response

Returns zero or more `artifactVersion` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getArtifactVersions [optionals]`

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. Argument type: String

Positional arguments

None

Response

Returns 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
```

Back to Top

getRepositories

Retrieves all artifact repository objects known to the ElectricFlow server.

Arguments	Descriptions
`None`	–

Positional arguments

None

Response

Returns 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. Argument type: String

Positional arguments

`repositoryName`

Response

Returns 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

getRetrievedArtifacts

Retrieves artifacts during a job.

You must specify an `jobId`.

Arguments	Descriptions
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`componentName`	(Optional) Name of the component. Argument type: String
`resourceName`	(Optional) Name of the resource. Argument type: String

Positional arguments

`jobId`

Response

Returns retrieved artifacts from the job.

ec-perl

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

Example

```
$cmdr->getRetrievedArtifacts("4fa765dd-73f1-11e3-b67e-b0a420524153", {componentName
=> "WAR files"});
```

ectool

syntax:`ectool getRetrievedArtifacts <jobId> [optionals]`

Example

```
ectool getRetrievedArtifacts "4fa765dd-73f1-11e3-b67e-b0a420524153" --componentName
"WAR files"
```

Back to Top

modifyArtifact

Modifies an existing artifact.

You must specify an `artifactName`.

Arguments	Descriptions
`artifactName`	The name of the artifact to modify. Argument type: String

Arguments	Descriptions
`artifactVersionNameTemplate`	(Optional) 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). Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String

Positional arguments

`artifactName`

Response

Returns a modified artifact element.

ec-perl

syntax: `$cmdr->modifyArtifact(<artifactName>, {<optionals>});`

Example

```
$cmdr->modifyArtifact("thirdParty-SDK", {description => "contains artifact versions
for SDK"});
```

ectool

syntax: `ectool modifyArtifact <artifactName> [optionals]`

Example

```
ectool modifyArtifact "thirdParty-SDK" --description "contains artifact versions fo
r 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 ElectricFlow server interprets either name form correctly. Argument type: String
`artifactVersionState`	(Optional) The state of the artifact version. <`publishing|available|unavailable`>. Argument type: ArtifactVersionState
`dependentArtifactVersions`	(Optional) 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. Argument type: Collection
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`newName`	(Optional) New name for this artifact version. Argument type: String
`removeAllDependentArtifactVersions`	(Optional) <*Boolean flag* - `0|1|true|false`> The default is `0` or `false` where dependencies are not removed." If this argument is set to `1` or `true`, all dependent artifacts will be removed from this artifact version. Subsequent "retrieves" will no longer retrieve dependent artifacts for this artifact version. Argument type: Boolean
`repositoryName`	(Optional) The name of the artifact repository. Argument type: String

Positional arguments

`artifactVersionName`

Response

Returns a modified artifact version element.

ec-perl

syntax: `$cmdr->modifyArtifactVersion(<artifactVersionName>, {<optionals>});`

Example

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

ectool

syntax: `ectool modifyArtifactVersion <artifactVersionName> [optionals]`

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
`repositoryName`	The name of the artifact repository. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`newName`	(Optional) New name of the repository. Argument type: String
`repositoryDisabled`	(Optional) <*Boolean flag* - `0|1|true|false`> This argument marks the repository as enabled (`0` or `false`) or disabled (`1` or `true`). If you do not specify this, the state of the repository is unchanged. Argument type: Boolean

Arguments	Descriptions
`url`	(Optional) The URL used to communicate with the artifact repository. Argument type: String
`zoneName`	(Optional) The name of the zone where this repository resides. Argument type: String

Positional arguments

`repositoryName`

Response

Returns a modified `repository` element.

ec-perl

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

Example

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

ectool

syntax: `ectool modifyRepository <repositoryName> [optionals]`

Example

```
ectool modifyRepository "cmdrRepository" --newName "flowRepository"
```

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 ElectricFlow 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. Argument type: String
`beforeRepositoryName`	(Optional) Moves this repository (`repositoryName`) to a place before the name specified by this option. If omitted `repositoryName` is moved to the end. Argument type: String

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> [optionals]`

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**
`artifactName`	The name of an artifact. Argument type: String
`artifactKey`	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String
`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. Argument type: Boolean

Arguments	Descriptions
`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. Argument type: Collection
`description`	A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`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. Argument type: Collection
`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. Argument type: Boolean
`fromDirectory`	The directory containing files to publish as the artifact version. A subset of files can be published based on `includePatterns` and `excludePatterns`. Argument type: String
`groupId`	A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String

Arguments	Descriptions
`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. Argument type: Collection
`repositoryName`	The name of the artifact repository where you want to publish. Argument type: String

`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 the examples below.
	Argument type: String

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

Returns 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 [optionals]`

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 ElectricFlow server interprets either name form correctly. Argument type: String
`dependentArtifactVersions`	(Optional) 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. Argument type: Collection

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> [optionals]`

Example

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

Back to Top

resolveRoute

Resolves the network route to an artifact repository.

You must specify the `toRepositoryName`.

Arguments	Descriptions
`toRepositoryName`	Name of the artifact repository. Argument type: String
`fromAgentId`	(Optional) Identifier of the agent requesting the route to a destination agent or artifact repository. Argument type: Long
`fromResourceName`	(Optional) Name of the resource requesting the route to a destination agent or artifact repository. Argument type: String

Positional arguments

`toRepositoryName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->resolveRoute (<toRepositoryName>, {<optionals>});`

Examples

```
$cmdr->resolveRoute("Web Server", {fromResourceName => "Agent 2"});
```

ectool

syntax: `ectool resolveRoute <toRepositoryName> [optionals]`

Example

```
ectool resolveRoute "Web Server" --fromResourceName "Agent 2"
```

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. Argument type: String
`artifactName`	The name of the artifact. Argument type: String
`artifactVersionName`	The name of the artifact version. Argument type: String
`cacheDirectory`	The directory where the artifact version is stored. **Note:** The artifact version files are stored in a subdirectory under this cache directory. Argument type: String

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 ElectricFlow 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 ElectricFlow 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
	Argument type: Collection
`groupId`	A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String
`includeDependents`	*<Boolean flag -* `0|1|true|false`> The default is `1` or `true`. If the value is `1` or `true`: • The artifact and its dependents are retrieved. • The published artifact version includes the artifact's dependents, such as a list of one or more artifact versions. • The dependent artifact versions are stored in a subdirectory under the `cacheDirectory` or if `toDirectory` is specified, under the `oDirectory/ec_dependent_artifacts` directory. If the value is `0` or `false`, only the artifact is retrieved. The artifact version does not include the dependent artifacts. Argument type: Boolean
`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. Argument type: String

Arguments	Descriptions
`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. Argument type: String
`retryNumber`	Number of retry attempts for the operation. The default is 1. The time between retry attempts is 20 seconds. Argument type: Integer
`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. Argument type: String
`versionRange`	The range of versions to search. Version range syntax is standard number interval notation. `()` marks exclusive ranges and `[]` marks inclusive ranges. Argument type: String

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 [optionals]`

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

updateArtifactVersion

Updates an artifact version by adding or replacing one or more files in the existing file and publishes the result as a new artifact version to an artifact repository.

Note: This API wraps the "update" 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 update.

Arguments	Descriptions
`artifactKey`	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String
`artifactName`	The name of the artifact. Argument type: String
`description`	A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String

Arguments	Descriptions
`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. Argument type: Collection
`followSymlinks`	<*Boolean flag* - `0|1|true|false`> The default is `1` or `true`. If `1` or `true`, ElectricFlow follows symbolic links and record the target file contents with the symbolic link name in the artifact. If `0` or `false`, ElectricFlow records the symbolic link as a symbolic link. Following symbolic links causes the publish API to remain compatible with previous releases. Argument type: Boolean
`fromDirectory`	The directory containing files to publish as the artifact version. A subset of files can be published based on `includePatterns` and `excludePatterns`. Argument type: String
`groupId`	A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, spaces, underscores, hyphens, and periods. Argument type: String
`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. Argument type: Collection

`newVersion`	Unique identifier for the new 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.
	
	Argument type: String

Arguments	Descriptions
`path`	The path of the original artifact under which one or more files will be added or replaced. The default path is the root. Argument type: String

`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.
	
	Argument type: String

Positional arguments

None

Response

Publishes a new artifact version to an artifact repository.

ec-perl

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

Examples

```
# Update the current myGroup:myKey artifact version to version 1.0.0-55.
$cmdr->updateArtifactVersion({artifactName => "myGroup:myKey",
                              newVersion => "1.0.0-55"});
```

ectool

syntax: `ectool updateArtifactVersion [optionals]`

Example

```
ectool updateArtifactVersion --artifactName "myGroup:myKey" --newVersion "1.0.0-55"
```

Back to Top

API Commands – Change History

revert on page 149

searchEntityChange on page 150

getDeploymentHistoryItems

Retrieves all the deployment history items for a specific environment.

You must specify `projectName` .

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`applicationName`	(Optional) The application that owns the deployment history item. Argument type: String
`environmentName`	(Optional) The name of the environment where the application runs. Argument type: String
`environmentProjectName`	(Optional) The name for the project to which the environment or environment template belongs. Argument type: String
`latest`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, only the latest deployment information is returned. Argument type: Boolean
`processName`	(Optional) The process that owns the deployment history item. Argument type: String
`snapshotName`	(Optional) The snapshot that owns the deployment history item. Argument type: String

Positional arguments

`projectName`

Response

Returns zero or more deployment history items.

ec-perl

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

Example

```
$ec->getSnapshot ("Tutorials", {applicationName => "Deploy"});
```

ectool

syntax: `ectool getSnapshot <projectName> [optionals]`

Example

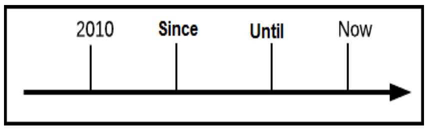
```
ectool getSnapshot "Tutorials" --applicationName "Deploy"
```

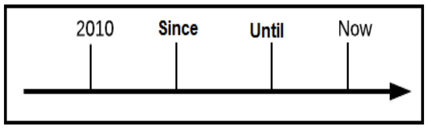
Back to Top

getEntityChange

Retrieves the entity changes.

You must specify `entityId`, `entityPath`, or `entityType`.

Arguments	Descriptions
`entityId`	Entity ID. Argument Type: String
`entityPath`	Path to the entity. Argument Type: String
`entityType`	Type of entity. Argument Type: String
`modifiedBy`	(Optional) Login ID of the user who modified the object. Argument Type: String
`timeSince`	(Optional) Changes since this time, the start time for changes. This is the time line:  Argument Type: Long

Arguments	Descriptions
`timeUntil`	(Optional) Changes up to this time, the end time for changes. If this argument is not specified, the default is *Now*. This is the time line:  Argument Type: Long

Positional arguments

`entityId`, `entityPath`, or `entityType`

Response

Returns entity changes during the time interval between `timeSince` and `timeUntil`.

ec-perl

Enter one of these commands:

syntax: `$<object>->getEntityChange(<entityId>, {<optionals>});`

syntax: `$<object>->getEntityChange(<entityPath>, {<optionals>});`

syntax: `$<object>->getEntityChange(<entityType>, {<optionals>});`

Example

If the `entityType` is component:

```
$ec->getEntityChange("WAR file", {timeUntil => 1600});
```

ectool

Enter one of these commands:

syntax:`ectool getEntityChange <entityId> [optionals]`

syntax:`ectool getEntityChange <entityPath> [optionals]`

syntax:`ectool getEntityChange <entityType> [optionals]`

Example

If the `entityType` is component:

```
ectool getEntityChange "WAR file" --timeUntil 1600
```

Back to Top

getEntityChangeDetails

Retrieves the differences between entities.

You must specify `entityId`, `entityType`, and `revisionNumber`.

Note: When ElectricFlow exports entity changes in XML, as well as listing the `ec_change_history_revision id` as `changeHistoryRevisionId`, it now also lists this as `revisionNumber`.

Arguments	Descriptions
`entityId`	The entity ID. Argument type: UUID
`entityType`	The entity type. Argument type: String
`revisionNumber`	The revision number of the entity. Argument type: Integer

Positional arguments

`entityId`, `entityType`, `revisionNumber`

ResponseDetails.

Returns an `entityChange` element.

ec-perl

syntax: `$<object>->getEntityChangeDetails (<entityId>, <entityType>, <revisionNumber>);`

Example

```
$ec->getEntityChangeDetails("4fa914dd-73f1-11e3-b67e-b0a420524153", "Process", "4")
;
```

ectool

syntax: `ectool getEntityChangeDetails <entityId> <entityType> <revisionNumber>`

Example

```
ectool getSnapshots "4fa914dd-73f1-11e3-b67e-b0a420524153" "Process" "4"
```

Back to Top

pruneChangeHistory

Prunes obsolete-for-days data from the Change History tables.

You must enter `daysToKeep`.

Arguments	Descriptions
`daysToKeep`	Number of days of Change History data to keep. The minimum is 7. Argument type: Long
`forcePruneAll`	(Optional) Use this argument with caution. It is used most often for testing. It overrides the specified `daysToKeep` value and prunes the entire Change History, keeping nothing discardable. The `forcePruneAll` value = <*Boolean flag* -`0|1|true|false`>. The default is `0` or `false`. Argument type: Boolean

Positional arguments

`daysToKeep`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->pruneChangeHistory (<daysToKeep>, {<optionals>});`

Example

```
$ec->pruneChangeHistory (14, {forcePruneAll => 0});
```

ectool

syntax:`ectool pruneChangeHistory <daysToKeep> [optionals]`

Example

```
ectool pruneChangeHistory 14 --forcePruneAll 0
```

Back to Top

revert

Reverts the state of the object to a previous state.

You must enter `objectID`, `objectType`, and `revisionNumber`.

You must have the *Read*, *Modify*, and *Execute* permissions to access to this API call.

Note: When ElectricFlow exports entity changes in XML, as well as listing the `ec_change_history_revision id` as `changeHistoryRevisionId`, it now also lists this as `revisionNumber`.

Arguments	Descriptions
`objectId`	Object ID. Argument type: UUID
`objectType`	Object type. Argument Type: String
`revisionNumber`	Revision number of the object. Argument Type: Integer

Positional arguments

`objectID`, `objectType`, `revisionNumber`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->revert (<objectID>, <objectType>, <revisionNumber>);`

Example

```
$ec->revert ("4fa914dd-73f1-11e3-b67e-b0a420524153", "property", 3);
```

ectool

syntax:`ectool revert <objectID> <objectType> <revisionNumber>`

Example

```
ectool revert "4fa914dd-73f1-11e3-b67e-b0a420524153" "property" 3
```

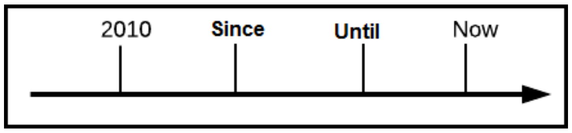
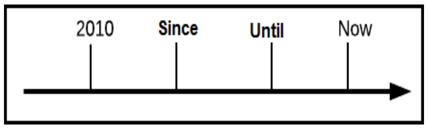
Back to Top

searchEntityChange

Searches for entity changes.

You must enter the `entityId`, `entityPath`, or `entityType`.

Arguments	Descriptions
`entityId`	The entity ID. Argument Type: String
`entityPath`	Path to the entity. Argument Type: String

Arguments	Descriptions
`entityType`	Type of entity. Argument Type: String
`modifiedBy`	(Optional) The login ID of the user who modified the object. Argument Type: String
`timeSince`	(Optional) Start of the time interval for changes. ElectricFlow searches for changes since this time. This is the time line:  Argument type: Long
`timeUntil`	(Optional) End of the time interval for changes. ElectricFlow searches for changes up to this time. If this argument is not specified, the default is *Now*. This is the time line:  Argument Type: Long

Positional arguments

`entityId`, `entityPath`, or `entityType`

Response

Returns entity changes during the time interval.

ec-perl

Enter one of these commands:

syntax: `$<object>->searchEntityChange (<entityId>, {<optionals>});`

syntax: `$<object>->searchEntityChange (<entityPath>, {<optionals>});`

syntax: `$<object>->searchEntityChange (<entityType>, {<optionals>});`

Example

If the `entityType` is component:

```
$ec->searchEntityChange("component", {timeUntil => 1600});
```

ectool

Enter one of these commands:

syntax: `ectool searchEntityChange <entityId> [optionals]`

syntax: `ectool searchEntityChange <entityPath> [optionals]`

syntax: `ectool searchEntityChange <entityType> [optionals]`

Example

If the `entityType` is component:

```
ectool searchEntityChange "component" --timeUntil 1600
```

Back to Top

API Commands - Components

addComponentToApplicationTier

Adds the specified component to the specified application tier.

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

Arguments	Descriptions
projectName	Name for the project that must be unique among all projects. Argument Type: String
applicationName	Name of the application that must be unique among all projects. Argument Type: String
applicationTierName	Name of the tier that must be unique within the application. Argument Type: String
componentName	Name of the component. Argument Type: String
componentProjectName	(Optional) Name of the project that contains the component. Argument Type: String

Positional arguments

`projectName`, `applicationName`, `applicationTierName`, `componentName`

Response

Returns the component and application tier elements.

ec-perl

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

Example

```
$ec->addComponentToApplicationTier("default", "Take snapshot", "Web server", "VC
Scomponent");
```

ectool

syntax: `ectool addComponentToApplicationTier <projectName> <applicationName> <applicationTierName> <componentName> [optionals]`

Example

```
ectool addComponentToApplicationTier "default" "Take snapshot" "Web server" "VCS
component"
```

Back to Top

copyComponent

Creates a new component based on an existing one.

You must specify the `projectName`, `componentName`, and `newComponentName`.

Arguments	Descriptions
projectName	The name for the project that must be unique among all projects. Argument Type: String
componentName	The name of the component. Argument Type: String
newComponentName	The name of the new component. Argument Type: String
applicationTierName	(Optional) The name of the tier that must be unique within the application. Argument Type: String
description	(Optional) Comment text describing this object. It is not interpreted by ElectricFlow. Argument Type: String
fromApplicationName	(Optional) The name of source application. Argument Type: String
toApplicationName	(Optional) The name of source application. Argument Type: String

Positional arguments

`projectName`, `componentName` , and `newComponentName`

Response

Returns the new component.

ec-perl

syntax: `$<object>->copyComponent(<projectName>, <componentName>, <newComponentName>, {<optionals>});`

Example

```
$ec->copyComponent("Default", "WAR file", "New WAR file", {applicationTierName =
> "Web Server Config"});
```

ectool

syntax: `ectool copyComponent <projectName> <componentName> <newComponentName> [optionals]`

Example:

```
ectool copyComponent "Default" "WAR file" "New WAR file" --applicationTierName "
Web Server Config"
```

Back to Top

createComponent

Creates a new component for a project.

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

Arguments	Descriptions
projectName	The name for the project that must be unique among all projects. Argument Type: String
componentName	The name of the component. Argument Type: String
actualParameters	(Optional) The parameters passed as arguments to the application component. Argument Type: Map
applicationName	(Optional) The name of the application where the component is defined. Argument Type: String
credentialName	(Optional) The name of the credential to attach to the component. Argument Type: String
description	(Optional) Comment text describing this object. It is not interpreted by ElectricFlow. Argument Type: String
pluginKey	(Optional) The key of the plugin. Argument Type: String
pluginName	(Optional) The name of the plugin. Argument Type: String
pluginParameters	(Optional) The list of the plugin parameters. Argument Type: Map
reference	(Optional) <*Boolean flag -* `0|1|true|false`> If `1` or `true`, a reference of the component is created. If `0` or `false`, a copy of the component is created. Argument type: Boolean
sourceApplicationName	(Optional) The name of source application. Argument Type: String

Arguments	Descriptions
sourceComponentName	(Optional) The name of new component. Argument Type: String
sourceProjectName	(Optional) The name of source project. Argument Type: String

Positional arguments

`projectName`, `componentName`

Usage Guidelines:

- To create a new component, use `pluginKey` or `pluginName`.

  ectool example: `--pluginKey` or `--pluginName`

- To create an application component by copying a master component, use `applicationName`, `sourceComponentName`, `sourceProjectName`, and `reference` = `0`.

  ectool example: `--applicationName, --sourceComponentName, --sourceProjectName, --reference 0`

- To create a master component by copying another master component, use `sourceComponentName`, `sourceProjectName`,and `reference` = `0`.

  ectool example: `--sourceComponentName, --sourceProjectName, --reference 0`

- To create an application component by copying another application component, use `applicationName`, `sourceComponentName`, `sourceApplicationName`, `sourceProjectName`, and `reference` = `0`.

  ectool example: `--applicationName, --sourceComponentName, --sourceApplicationName, --sourceProjectName, --reference 0`

- To create a master component from an application component, use `sourceComponentName`, `sourceApplicationName`, `sourceProjectName`, and `reference` = `0`.

  ectool example: `--sourceComponentName, --sourceApplicationName, --sourceProjectName, --reference 0`

- To create a reference of the master component, use `applicationName`, `sourceComponentName`, `sourceProjectName`, and `reference` = `1`.

  ectool example: `--applicationName, --sourceComponentName, --sourceProjectName, --reference 1`

Response

Returns a version-controlled component element.

ec-perl

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

Example

To create a new component:

```
$ec->createComponent("Default", "Cleanup DB", {pluginName => "EC-Maven"});
```

To create an application component by copying a master component

```
$ec->createComponent("Default", "Cleanup DB", {applicationName => "Deploy",
sourceComponentName => "Backup DB", sourceProjectName => "Archive", reference =>
0});
```

ectool

syntax: `ectool createComponent <projectName> <componentName> [optionals]`

Example

To create a new component:

```
ectool createComponent "Default" "Cleanup DB" --pluginName "EC-Maven"
```

To create an application component by copying a master component:

```
ectool createComponent "Default" "Cleanup DB" --applicationName "Deploy" --sourc
eComponentName "Backup DB"
--sourceProjectName "Archive" --reference 0
```

Back to Top

deleteComponent

Deletes a component.

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

Arguments	Descriptions
projectName	Name for the project that 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

Positional arguments

`projectName`, `componentName`

Response

None or a status OK message.

ec-perl

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

Example

```
$ec->deleteComponent("Default", "VCScomponent", {applicationName => "Deploy"});
```

ectool

syntax: `ectool deleteComponent <projectName> <componentName> [optionals]`

Example

```
ectool deleteComponent "Default" "VCScomponent"  --applicationName "Deploy"
```

Back to Top

getComponent

Retrieves a component by name.

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

Arguments	Descriptions
projectName	Name for the project that must be unique among all projects. Argument Type: String
componentName	Name of the component. Argument Type: String
applicationEntityRevisionId	(Optional) The revision ID of the versioned object. Argument type: UUID
applicationName	(Optional) Name of an application to which this component is scoped. Argument Type: String

Positional arguments

`projectName, componentName`

Response

Retrieves the component element.

ec-perl

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

Example

```
$ec->getComponent("Default", "WAR file", {applicationEntityRevisionId => "4fa765
dd-73f1-11e3-b67e-b0a420524153"});
```

ectool

syntax: `ectool getComponent <projectName> <componentName> [optionals]`

Example

```
ectool getComponent "Default" "WAR file" --applicationEntityRevisionId "4fa765d
d-73f1-11e3-b67e-b0a420524153"
```

Back to Top

getComponents

Retrieves all components in a project.

You must specify the `projectName`.

Arguments	Descriptions
projectName	Name for the project that must be unique among all projects. Argument Type: String
applicationEntityRevisionId	(Optional) The revision ID of the versioned object. Argument type: UUID
applicationName	(Optional) Name of the application. You can search for components scoped to an application. Argument Type: String

Positional arguments

`projectName`

Response

Returns zero or more component elements.

ec-perl

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

Example

```
$ec->getComponents("Default", {applicationEntityRevisionId => "4fa765dd-73f1-11e
3-b67e-b0a420524153"});
```

ectool

syntax: `ectool getComponents <projectName> [optionals]`

Example

```
ectool getComponents "Default" --applicationEntityRevisionId "4fa765dd-73f1-11e
3-b67e-b0a420524153"
```

Back to Top

getComponentsinApplicationTier

Retrieves the list of components in an application tier.

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

Arguments	Descriptions
projectName	Name for the project that must be unique among all projects. Argument Type: String
applicationName	Name of the application that must be unique among all projects. Argument Type: String
applicationTierName	Name of the tier that must be unique within the application. Argument Type: String
applicationEntityRevisionId	(Optional) The revision ID of the versioned object. Argument type: UUID
includeArtifactDetail	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, the artifact name and version are returned as part of component response. Argument type: Boolean

Positional arguments

`projectName, applicationName, applicationTierName`

Response

Returns zero or more component elements in the application tier.

ec-perl

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

Example:

```
$ec->getComponentsInApplicationTier("Default", "Deploy snapshot", "AWS", {applic
ationEntityRevisionId => "4fa765dd-73f1-11e3-b67e-b0a420524153"});
```

ectool

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

Example:

```
ectool getComponentsInApplicationTier "Default" "Deploy snapshot" "AWS" --applic
ationEntityRevisionId 4fa765dd-73f1-11e3-b67e-b0a420524153
```

Back to Top

modifyComponent

Modifies an existing component.

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

Arguments	Descriptions
projectName	Name for the project that must be unique among all projects. Argument Type: String
componentName	Name of the component. Argument Type: String
actualParameters	(Optional) Parameters passed as arguments to the application component. Argument Type: Map
applicationName	(Optional) Name of an application to which this component is scoped. Argument Type: String
clearActualParameters	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, all actual parameters will be removed. Argument type: Boolean
credentialName	(Optional) Name of the credential to attach to this component. Argument Type: String
description	(Optional) Comment text describing this component. It is not interpreted by ElectricFlow. Argument Type: String
newName	(Optional) New name of the component. Argument Type: String
pluginKey	(Optional) Key for the plugin. Argument Type: String
pluginName	(Optional) Name of the plugin. Argument Type: String
pluginParameters	(Optional) List of the plugin parameters Argument Type: Map

Positional arguments

`projectName`, `componentName`

Response

Returns an updated component element.

ec-perl

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

Example

```
$ec->modifyComponent("Default", "Web Server", {credentialName => "cred1", newNam
e => "Master Web Server"});
```

ectool

syntax: `ectool modifyComponent <projectName> <componentName> [optionals]`

Example

```
ectool modifyComponent "Default" "Web Server" --credentialName cred1 --newName "
Master Web Server"
```

Back to Top

removeComponentFromApplicationTier

Removes the specified component from an application tier.

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

Arguments	Descriptions
projectName	Name for the project that must be unique among all projects. Argument Type: String
applicationName	Name of the application that must be unique among all projects. Argument Type: String
applicationTierName	Name of the tier that must be unique within the application. Argument Type: String
componentName	Name of component. Argument Type: String
componentProjectName	(Optional) Name of the project that contains the component. Argument Type: String

Positional arguments

`projectName`, `applicationName`, `applicationTierName`, `componentName`

Response

None or a status OK message.

ec-perl

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

Example

```
$ec->removeComponentFromApplicationTier("Default", "Deploy", "Config", "WAR fil
e", {componentProjectName => "Server Setup"});
```

ectool

syntax: `ectool removeComponentFromApplicationTier <projectName> <applicationName> <applicationTierName> [optionals] <componentName> [optionals]`

Example

```
ectool removeComponentFromApplicationTier "Default" "Deploy" "Config" "WAR file"
--componentProjectName "Server Setup"
```

Back to Top

API Commands - Credential Management

attachCredential

Attaches a credential to an object, such as 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 an object.

Arguments	Descriptions
`projectName`	The name of the project that must be must be unique among all projects. Argument type: String

Arguments	Descriptions
`credentialName`	Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`Locator arguments:`	
`applicationName`	The name of the application process to which the credential is attached. Argument type: String
`applicationProjectName`	The name of the project containing the application. If not specified, it defaults to the Release project name. Argument type: String
`componentName`	The name of the component to which the credential is attached when attaching a credential to a component, component process, or component process step. Argument type: String
`pipelineName`	The name of the pipeline when attaching a credential to a stage task.
`procedureName`	The name of a procedure when attaching a credential to a procedure or procedure step. Argument type: String
`processName`	(Optional) The name of a process when attaching a credential to a process or process step. Argument type: String
`processStepName`	The name of a process step when attaching a credential to a process step. Argument type: String
`releaseName`	The name of a Release when attaching a credential to a Release. Argument type: String

Arguments	Descriptions
`scheduleName`	The name of the schedule for running a procedure or process in the "named" project when attaching a credential to the schedule. Argument type: String
`stageName`	The name of the stage when attaching a credential to a task. Argument type: String
`stateDefinitionName`	The name of the workflow state definition when attaching a credential to a state definition. Argument type: String
`stepName`	A step name in a procedure or process in the "named" project when attaching a credential to a procedure step. Argument type: String
`taskName`	The name of the task when attaching a credential to the task. Argument type: String
`workflowDefinitionName`	The name of the workflow when attaching a credential to a state definition. Argument type: String

Positional arguments

`projectName`, `credentialName`, and locator arguments to identify an object.

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->attachCredential("Default", "QA User", {procedureName =>
   "Run Build", stepName=>"Get Resources"});
```

ectool

syntax: `ectool attachCredential <projectName> <credentialName> [optionals]`

Example

```
ectool attachCredential "Default" "QA User" --procedureName "Run Build" --stepName
"Get Resources"
```

Back to Top

createCredential

Creates a new credential for a project.

You must specify a `projectName`, `credentialName`, `username`, and `password`.

Arguments	Descriptions
`projectName`	The name of the project where the credential will be stored. The name must be unique within all projects. Argument type: String
`credentialName`	The name of the credential. Argument type: String
`userName`	The user name of the credential. Argument type: String
`password`	The password of the credential. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`passwordRecoveryAllowed`	(Optional) <*Boolean flag* - `0|1|true|false`>–If `1` or `true`, the password can be recovered by running `getFullCredential` from a job step. Argument type: Boolean

Positional arguments

`projectName, credentialName, userName, password`

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->createCredential("Default", "QA Deploy", "QA", "abc123", {description => "Us
e during preproduction"});
```

ectool

syntax: `ectool createCredential <projectName> <credentialName> <userName> <password> [optionals]`

Example

```
ectool createCredential "Default" "QA Deploy" "QA" "abc123" --description "Use for
preproduction"
```

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. The project name must be unique among all projects. Argument type: String
`credentialName`	Name of the credential in one of these 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. Argument type: String

Positional arguments

`projectName, credentialName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

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

Example

```
ectool deleteCredential "Default" "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
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`credentialName`	Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`applicationName`	(Optional) The name of the application process with the credential that you want to detach. Argument type: String
`applicationProjectName`	(Optional) The name of the project containing specified application. If this is not specified, the default is the Release project name. Argument type: String
`componentName`	(Optional) The name of the component or component process with the credential that you want to detach.
`pipelineName`	(Optional) The name of the pipeline when a credential attached to a stage task. Argument type: String
`procedureName`	(Optional) The name of the procedure with the credential that you want to detach. Argument type: String

Arguments	Descriptions
`processName`	(Optional) The name of the process with the credential that you want to detach. Argument type: String
`processStepName`	(Optional) The name of the process step with the credential that you want to detach. Argument type: String
`releaseName`	(Optional) The name of the Release defined by a pipeline to which a credential attached. Argument type: String
`scheduleName`	(Optional) The name of the schedule where this credential is attached. Argument type: String
`stageName`	(Optional) The name of the stage when a credential is attached to a stage task. Argument type: String
`stateDefinitionName`	(Optional) The name of the workflow state definition when a credential is attached to a state definition. Argument type: String
`stepName`	(Optional) A step name in a procedure or process in the "named" project. Argument type: String
`taskName`	(Optional) The name of the task when a credential is attached to a task. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition when a credential is attached to a state definition. Argument type: String

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("Default", "Preflight User",
                         {procedureName => "Run Build",
                               stepName => "Get Sources"});

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

ectool

syntax: `ectool detachCredential <projectName> <credentialName> [optionals]`

Examples

```
ectool detachCredential "Default" "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
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`credentialName`	Name of the credential in one of these 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. Argument type: String

Positional arguments

`projectName, credentialName`

Response

One `credential` element.

ec-perl

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

Example

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

ectool

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

Example

```
ectool getCredential "QA Runs" "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 that must be unique among all projects. Argument type: String
`usableOnly`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to *1* or `true`, only those credentials that the currently logged-in user has execute privileges for will be returned. Argument type: Boolean

Positional arguments

`projectName`

Response

Zero or more `credential` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getCredentials <projectName> [optionals]`

Example

```
ectool getCredentials "Default" --usableOnly 1
```

Back to Top

getFullCredential

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

You must specify the `credentialName` and `jobStepId`.

Arguments	Descriptions
`credentialName`	Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`jobStepId`	The unique identifier for the job step that is used to make a project association. Argument type: UUID

Positional arguments

`credentialName`, `jobStepId`

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>, <jobStepId>);`

Example

```
# Returns an xPath object containing the password.
my $xpath = $cmdr->getFullCredential("myCred", "4fa765dd-73f1-11e3-b67e-b0a42052415
3");


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

ectool

syntax: `ectool getFullCredential <credentialName> <jobStepId>`

Example

```
ectool getFullCredential "myCred" "4fa765dd-73f1-11e3-b67e-b0a420524153"
```

Back to Top

modifyCredential

Modifies an existing credential.

You must specify `projectName` and `credentialName`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`credentialName`	Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`newName`	(Optional) New name of the credential. Argument type: String
`password`	(Optional) The password for the credential. It can also be a certificate or other chunk of data. Argument type: String
`passwordRecoveryAllowed`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, the password can be recovered by running `getFullCredential` from a job step. Argument type: Boolean
`userName`	The name of the user for this credential. Argument type: String

Positional arguments

`projectName, credentialName`

Response

Returns an updated credential element.

ec-perl

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

Example

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

ectool

syntax: `ectool modifyCredential <projectName> <credentialName> [optionals]`

Example

```
ectool modifyCredential "Default" "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
```

Back to Top

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 ElectricFlow server after configuring the new database you want to use.

ElectricFlow assigns default values to the following three arguments that are derived from information you enter 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 override ElectricFlow 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 chooses an appropriate dialect based on the `databaseType`). Argument Type: String
`customDatabaseDriver`	Class name of the JDBC driver (*advanced use only*. The server chooses an appropriate driver based on the `databaseType`). Argument Type: String
`customDatabaseUrl`	The JDBC to use (*advanced use only*. The server composes an appropriate URL). Argument Type: String
`databaseName`	The name of the database that you want the ElectricFlow server to use. The default is *commander*. Argument Type: String
`databaseType`	The type of database that you want the ElectricFlow server to use. Supported database types are: `<builtin|mysql|sqlserver|oracle>`. The default is *builtin*. Argument Type: DatabaseType

Arguments	Descriptions
`hostName`	The domain name or IP address of the host server machine where the database is running. Argument Type: String
`ignorePasskeyMismatch`	<*Boolean flag -* `0|1|true|false`> If `true`, ignore a passkey fingerprint mismatch between the current passkey file and the database configuration and discard the stored credentials. **Note:** This action discards all saved passwords. Argument Type: Boolean
`ignoreServerMismatch`	<*Boolean flag -* `0|1|true|false`> If `true`, ignore a host name mismatch between the current server and the database configuration where the server previously started. Argument Type: Boolean
`password`	The password required to access the database. `setDatabaseConfiguration` does not allow a passwordless database user. Make sure the database user has a password. Argument Type: String
`port`	The port number used to access the database. The default is the server port default. Argument Type: String
`preserveSessions`	<*Boolean flag -* `0|1|true|false`> When a host name mismatch between the current server and the database configuration occurs, the default behavior is to invalidate all sessions. If this argument is set to true, all sessions are preserved and the server can reconnect to running jobs. This option is used in combination with `ignoreServerMismatch`. Argument Type: Boolean
`userName`	The name of the user required to access the database. Argument Type: String

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> [optionals]`

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

validateDatabase

Performs consistency checks on the database.

Arguments	Descriptions
`options`	Comma-separated list of options that specify the aspects of the database to validate. Argument Type: String

Positional arguments

None

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool validateDatabase [optionals]`

Example

```
ectool validateDatabase
```

Back to Top

API Commands - Directory Provider Management

`createDirectoryProvider`

Creates a new Active Directory or LDAP directory provider.

You must specify a `providerName` .

Arguments	Descriptions
`providerName`	Name for a LDAP directory provider that must be unique. This human-readable name appears in the user interface to identify users and groups from this provider. Argument type: String
`allowNestedGroupsApprovers`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether users in nested LDAP groups should be allowed to approve a manual task when a parent LDAP group is assigned as a approver for the task and recursive traversal of group hierarchy is enabled for the directory provider. Argument type: Boolean
`commonGroupNameAttribute`	(Optional) 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. Argument type: String

Arguments	Descriptions
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`domainName`	(Optional) The domain name from which Active Directory servers are automatically discovered. Argument type: String
`emailAttribute`	(Optional) The attribute in an LDAP 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. Argument type: String
`enableGroups`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether or not to enable external groups for the directory provider. This argument defaults to "true". Argument type: Boolean
`fullUserNameAttribute`	(Optional) The attribute in a user record that contains the user's full name (first and last) to display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead. Argument type: String
`groupBase`	(Optional) The string is prepended to the `basedn` to construct the directory domain name (DN) that contains group records. Argument type: String
`groupMemberAttributes`	(Optional) 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 may be required. Argument type: String

Arguments	Descriptions
`groupMemberFilter`	(Optional) 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. Argument type: String
`groupNameAttribute`	(Optional) The attribute in a group record that contains the name of the group. Argument type: String
`groupSearchFilter`	(Optional) This LDAP query is performed in the context of the groups directory to enumerate group records. Argument type: String
`managerDn`	(Optional) The domain name (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. Argument type: String
`managerPassword`	(Optional) If the `managerDn` property is set, this password is used to authenticate the manager user. Argument type: String
`membershipAttribute`	(Optional) Attribute defined on an LDAP user or group entry used by the LDAP provider to specify the group membership. Argument type: String
`membershipFilter`	(Optional) LDAP filter to search for groups to which an LDAP user or group belongs. Argument type: String
`nestedGroupDepthLimit`	(Optional) Maximum number of group hierarchy levels that will be traversed for retrieving nested group membership information. Argument type: Integer

Arguments	Descriptions
`notifyUsersInNestedGroups`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether users in nested LDAP groups should be included when notifications for a parent LDAP group are sent and recursive traversal of group hierarchy is enabled for the directory provider. Argument type: Boolean
`providerType`	(Optional) Type string for a directory provider: `<ldap|activedirectory>` Argument type: ProviderType
`realm`	(Optional) 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 ElectricFlow server. For example, <*user*>@dir (where the realm is set to "dir"). Argument type: String
`traverseHierarchy`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether or not to enable recursive traversal of group hierarchy for nested group membership information. This argument defaults to "true". Argument type: Boolean
`url`	(Optional) 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`). Argument type: String
`useSSL`	(Optional) <*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. The default is "true". **Note:** Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server. Argument type: Boolean

Arguments	Descriptions
`userBase`	(Optional) This string is prepended to the `basedn` to construct the directory DN that contains user records. Argument type: String
`userNameAttribute`	(Optional) The attribute in a user record that contains the user's account name. Argument type: String
`userSearchFilter`	(Optional) 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. Argument type: String
`userSearchSubtree`	(Optional) <*Boolean flag* - `0|1|true|false`> If `1` or `true`, ElectricFlow recursively searches the subtree below the user base. Argument type: Boolean

Positional arguments

`providerName`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool createDirectoryProvider <providerName> {optionals]`

Example

```
ectool createDirectoryProvider AD3 --url "ldaps://pdc/dc=coname3.dc=com" --provider
Type 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 that you want to delete. Argument Type: String

Positional arguments

`providerName`

Response

None or a status OK message.

ec-perl

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

Example

```
$ec->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 that must be unique. Argument Type: String

Positional arguments

`providerName`

Response

One `directoryProvider` element.

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

ec-perl

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

Example

```
$ec>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: `$<object>->getDirectoryProviders();`

Example

```
$ec->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
`providerName`	The name of the directory provider that must be unique. Argument Type: String
`allowNestedGroupsApprovers`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether users in nested LDAP groups should be allowed to approve a manual task when a parent LDAP group is assigned as a approver for the task and recursive traversal of group hierarchy is enabled for the directory provider. Argument type: Boolean
`commonGroupNameAttribute`	(Optional) 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. Argument Type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. Argument Type: String
`domainName`	(Optional) The domain from which Active Directory servers are automatically discovered. Argument Type: String
`emailAttribute`	(Optional) 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. Argument Type: String
`enableGroups`	(Optional) <*Boolean flag* - `0|1|true|false`> This determines whether or not to enable external groups for the directory provider. Defaults to `true`. Argument Type: Boolean
`fullUserNameAttribute`	(Optional) 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. Argument Type: String

Arguments	Descriptions
`groupBase`	(Optional) This string is prepended to the `basedn` to construct the directory DN that contains group records. Argument Type: String
`groupMemberAttributes`	(Optional) 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. Argument Type: String
`groupMemberFilter`	(Optional) 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. Argument Type: String
`groupNameAttribute`	(Optional) The group record attribute that contains the name of the group. Argument Type: String
`groupSearchFilter`	(Optional) A filter name: this LDAP query is performed in the context of the groups directory to enumerate group records. Argument Type: String
`managerDn`	(Optional) 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. Argument Type: String
`managerPassword`	(Optional) If the `managerDn` property is set, this password is used to authenticate the manager user. Argument Type: String
`membershipAttribute`	(Optional) Attribute defined on an LDAP user or group entry used by the LDAP provider to specify the group membership. Argument type: String

Arguments	Descriptions
`membershipFilter`	(Optional) LDAP filter to search for groups to which an LDAP user or group belongs. Argument type: String
`nestedGroupDepthLimit`	(Optional) Maximum number of group hierarchy levels that will be traversed for retrieving nested group membership information. Argument type: Integer
`newName`	(Optional) New name of the directory provider. Argument Type: String
`notifyUsersInNestedGroups`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether users in nested LDAP groups should be included when notifications for a parent LDAP group are sent and recursive traversal of group hierarchy is enabled for the directory provider. Argument type: Boolean
`providerType`	(Optional) Type string for a directory provider: <`ldap|activedirectory`> . Argument Type: DirectoryType
`realm`	(Optional) 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 ElectricFlow server. For example, <*user*>@dir (where the `realm` is set to "dir"). Argument Type: String
`traverseHierarchy`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether or not to enable recursive traversal of group hierarchy for nested group membership information. This argument defaults to "true". Argument type: Boolean
`url`	(Optional) 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`). Argument Type: String

Arguments	Descriptions
`useSSL`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, SSL is used for communication. The default is `1` or `true`. Argument Type: Boolean
`userBase`	(Optional) This string is prepended to the `basedn` to construct the directory DN that contains user records. Argument Type: String
`userNameAttribute`	(Optional) The attribute in a user record that contains the user's account name. Argument Type: String
`userSearchFilter`	(Optional) 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 Argument Type: String .
`userSearchSubtree`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, ElectricFlow recursively searches the subtree below the user base. Argument Type: Boolean
`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". **Note:** Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server. Argument Type: Boolean

Positional arguments

`providerName`

Response

Returns a modified LDAP directory provider element.

ec-perl

syntax: `$<object>->modifyDirectoryProvider(<providerName>, {<optionals>});`

Example

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

ectool

syntax: `ectool modifyDirectoryProvider <providerName> [optionals]`

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 that must be unique. Argument Type: String
`beforeProviderName`	Moves this directory provider (`providerName`) to a place before the name specified by this option. If omitted, `providerName` is moved to the end. Argument Type: String

Positional arguments

`providerName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->moveDirectoryProvider(<providerName>, {<optionals>});`

Example

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

ectool

syntax: `ectool moveDirectoryProvider <providerName> [optionals]`

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
`userName`	The name of the user you are testing for this provider. Argument Type: String
`password`	The password for the user that you are testing for this provider. The command will prompt for the password if it is omitted. Argument Type: String
`allowNestedGroupsApprovers`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether users in nested LDAP groups should be allowed to approve a manual task when a parent LDAP group is assigned as a approver for the task and recursive traversal of group hierarchy is enabled for the directory provider. Argument type: Boolean
`commonGroupNameAttribute`	(Optional) 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. Argument Type: String
`domainName`	(Optional) The domain from which Active Directory servers are automatically discovered. Argument Type: String
`emailAttribute`	(Optional) 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. Argument Type: String
`enableGroups`	(Optional) <*Boolean flag* - `0|1|true|false`> Determines whether or not to enable external groups for the directory provider. Defaults to "true". Argument Type: Boolean
`fullUserNameAttribute`	(Optional) 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. Argument Type: String

Arguments	Descriptions
`groupBase`	(Optional) This string is prepended to the `basedn` to construct the directory DN that contains group records. Argument Type: String
`groupMemberAttributes`	(Optional) 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. Argument Type: String
`groupMemberFilter`	(Optional) 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. Argument Type: String
`groupNameAttribute`	(Optional) The group record attribute that contains the name of the group. Argument Type: String
`groupSearchFilter`	(Optional) This LDAP query is performed in the context of the groups directory to enumerate group records. Argument Type: String
`managerDn`	(Optional) 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. Argument Type: String
`managerPassword`	(Optional) If the `managerDn` property is set, this password is used to authenticate the manager user. Argument Type: String
`membershipAttribute`	(Optional) Attribute defined on an LDAP user or group entry used by the LDAP provider to specify the group membership. Argument type: String

Arguments	Descriptions
`membershipFilter`	(Optional) LDAP filter to search for groups to which an LDAP user or group belongs. Argument type: String
`nestedGroupDepthLimit`	(Optional) Maximum number of group hierarchy levels that will be traversed for retrieving nested group membership information. Argument type: Integer
`notifyUsersInNestedGroups`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether users in nested LDAP groups should be included when notifications for a parent LDAP group are sent and recursive traversal of group hierarchy is enabled for the directory provider. Argument type: Boolean
`providerType`	(Optional) Type string for a directory provider: `<ldap|activedirectory>`. Argument Type: DirectoryType
`realm`	(Optional) 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 ElectricFlow server. For example, <*user*>@dir (where the realm is set to "dir"). Argument Type: String
`traverseHierarchy`	(Optional) <*Boolean flag* -`0|1|true|false`> Determines whether or not to enable recursive traversal of group hierarchy for nested group membership information. This argument defaults to "true". Argument type: Boolean
`url`	(Optional) 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`). Argument Type: String

Arguments	Descriptions
`useDefaults`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, the default values will be used for all fields that are not specified. Argument Type: Boolean
`useSSL`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, SSL is used for communication. Argument Type: Boolean
`userBase`	(Optional) This string is prepended to the base DN to construct the directory DN that contains user records. Argument Type: String
`userNameAttribute`	(Optional) The attribute in a user record that contains the user's account name. Argument Type: String
`userSearchFilter`	(Optional) 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. Argument Type: String
`userSearchSubtree`	(Optional) <*Boolean flag* - `0|1|true|false`> If "true", recursively search the subtree below the user base. Argument Type: Boolean

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

Example

```
$ec->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 - Dynamic Environments

addResourcePoolToEnvironmentTier

Adds a resource pool to a specific environment tier. A resource pool is a named group of resources.

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

Arguments	Descriptions
`resourcePoolName`	Name of the resource pool that must be unique among all resource pools. Argument Type: String
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentName`	Name of the environment that must be unique among all environments. Argument Type: String
`environmentTierName`	Name of the environment tier that must be unique among all tiers for the environment. Argument Type: String

Arguments	Descriptions
`rollingDeployPhaseName`	(Optional) Name for the rolling deploy phase to be associated with the resource pool. Argument Type: String

Positional arguments

`resourcePoolName, projectName, environmentName, environmentTierName`

Response

Returns the resource pool and environment tier elements.

ec-perl

syntax: `$<object>->addResourcePoolToEnvironmentTier(<resourcePoolName>, <projectName>, <environmentName>, <environmentTierName>, {<optionals>});`

Example

```
$cmdr->addResourcePoolToEnvironmentTier("pool1", "Default", "Production", "Web Serv
er", {rollingDeployPhaseName => "Blue PROD"});
```

ectool

syntax: `ectool addResourcePoolToEnvironmentTier <resourcePoolName> <projectName> <environmentName> <environmentTierName> [optionals]`

Example

```
ectool addResourcePoolToEnvironmentTier "pool1" "Default" "Production" "Web Server"
--rollingDeployPhaseName "Blue PROD"
```

Back to Top

addResourceTemplateToEnvironmentTemplateTier

Adds a resource template to the specified environment template tier.

You must specify the `resourceTemplateName`, `projectName`, `environmentTemplateName`, and `environmentTemplateTierName`arguments.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String

Arguments	Descriptions
`resourceTemplateName`	Name of the resource template that must be unique among all resource templates. Argument Type: String
`environmentTemplateTierName`	Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String
`resourceCount`	(Optional) Number of resources to be spun from the resource template. Argument Type: Integer
`resourceTemplateProjectName`	(Optional) Name of the project to which the resource template belongs. Argument Type: String

Positional arguments

`projectName, environmentTemplateName, resourceTemplateName , environmentTemplateTierName`

Response

Returns the resource template and environment template tier elements.

ec-perl

syntax: `$<object>->addResourceTemplateToEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <resourceTemplateName>, <environmentTemplateTierName>, {<optionals>});`

Example

```
$ec->addResourceTemplateToEnvironmentTemplateTier("Default", "Production", "Test
station", "WebServer", {resourceCount => 4});
```

ectool

syntax: `ectool addResourceTemplateToEnvironmentTemplateTier <resourceTemplateName> <projectName> <environmentTemplateName> <environmentTemplateTierName> [optionals]`

Example

```
ectool addResourceTemplateToEnvironmentTemplateTier  "default" "Production" "Tes
t station" "WebServer" --resourceCount 4
```

Back to Top

addResourceToEnvironmentTemplateTier

Adds a resource to the specified environment template tier.

You must specify the `resourceName`, `projectName`, `environmentTemplateName`, and `environmentTemplateTierName`arguments.

Arguments	Descriptions
`resourceName`	Name of the resource that must be unique among all resources. Argument Type: String
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`environmentTemplateTierName`	Name for the environment template tier that must be unique among all tiers for the environment template. Argument Type: String

Positional arguments

`resourceName, projectName, environmentTemplateName, environmentTemplateTierName`

Response

Returns the resource and environment template tier elements.

ec-perl

syntax: `$<object>->addResourceToEnvironmentTemplateTier(<resourceName>, <projectName>, <environmentTemplateName>, <environmentTemplateTierName>);`

Example

```
$ec->addResourceToEnvironmentTemplateTier("Test station", "Default", "QA", "Tomc
at");
```

ectool

syntax: `ectool addResourceToEnvironmentTemplateTier <resourceName> <projectName> <environmentTemplateName> <environmentTemplateTierName>`

Example

```
ectool addResourceToEnvironmentTemplateTier "Test station" "Default" "QA" "Tomca
t"
```

Back to Top

createEnvironmentTemplate

Creates an environment template.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. This text is not interpreted by ElectricFlow. Argument type: String

Positional arguments

`projectName, environmentTemplateName`

Response

Returns an environment template object.

ec-perl

syntax: `$<object>->createEnvironmentTemplate(<projectName>, <environmentTemplateName>, {<optionals>});`

Example

```
$ec->createEnvironmentTemplate("Default", "PROD Server");
```

ectool

syntax: `ectool createEnvironmentTemplate <projectName> <environmentTemplateName> [optionals]`

Example

```
ectool createEnvironmentTemplate "Default" "PROD Server"
```

Back to Top

createEnvironmentTemplateTier

Creates a tier in an environment template.

You must specify the `projectName`, `environmentTemplateName`, and `environmentTemplateTierName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`environmentTemplateTierName`	Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. This text is not interpreted by the automation platform. Argument type: String
`resourceCount`	(Optional) The number of resources to be spun from a given resource template. Argument type: Integer
`resourceNames`	(Optional) List of resources to add to the environment tier. Argument type: Collection
`resourceTemplateName`	(Optional) Name for the resource template that must be unique among all resource templates. Argument type: String
`resourceTemplateProjectName`	(Optional) Name for the project to which the resource template belongs. Argument type: String

Positional arguments

`projectName, environmentTemplateName, environmentTemplateTierName`

Response

Returns an environment tier in an environment template.

ec-perl

syntax: `$<object>->createEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <environmentTemplateTierName>, {<optionals>});`

Example

```
$ec->createEnvironmentTemplateTier("Default", "PROD Server", "Web Services");
```

ectool

syntax: `ectool createEnvironmentTemplateTier <projectName> <environmentTemplateName> <environmentTemplateTierName> [optionals]`

Example

```
ectool createEnvironmentTemplateTier "Default" "PROD Server" "Web Services"
```

Back to Top

createEnvironmentTemplateTierMap

Creates an environment-template tier map for an application.

You must specify the `projectName`, `applicationName`, `environmentProjectName`, and `environmentTemplateName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application. Argument Type: String
`environmentProjectName`	Name for the project to which the environment belongs. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`applicationEntityRevisionId`	(Optional) Revision ID of the versioned object Argument Type: UUID
`tierMapName`	(Optional) Name of the tier map associated with the environment template. If you do not specify an tier map, ElectricFlow uses a tier map with a hyphenated-application-and-environment name. Argument type: String
`tierMappings`	(Optional) List of mappings between the application tiers and the environment template tiers. Argument Type: Map

Positional arguments

`projectName`, `applicationName`, `environmentProjectName`, `environmentTemplateName`

Response

Returns a tier map for the environment template.

ec-perl

syntax: `$<object>->createEnvironmentTemplateTierMap(<projectName>, <applicationName>, <environmentProjectName>, <environmentTemplateName>, {<optionals>});`

Example

```
$ec->createEnvironmentTemplateTierMap("Default", "Undeploy", "PROD", "Web Servic
es");
```

ectool

syntax: `ectool createEnvironmentTemplateTierMap <projectName> <applicationName> <environmentProjectName> <environmentTemplateName> [optionals]`

Example

```
ectool createEnvironmentTemplateTierMap "Default" "Undeploy" "PROD" "Web Service
s
```

Back to Top

createHook

Creates a hook in a resource template, which can have one or more hooks. A hook stores a reference to a procedure in an ElectricFlow project or plugin project. When a resource template is used to create a resource pool, these procedures are invoked.

You must specify the `hookName`.

Arguments	Descriptions
`hookName`	Name of the hook that must be unique among all hooks in the project. Argument Type: String
`broadcast`	(Optional) Broadcast flag Use this flag to broadcast the hook name in the project. The `broadcast` value = <*Boolean flag* -`0|1|true|false`>. Defaults to `true` or `1`. Argument type: Boolean

Arguments	Descriptions
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. This text is not interpreted by the automation platform. Argument type: String
`hookParameters`	(Optional) Parameters that are passed to the procedure. Argument type: Map
`hookType`	(Optional) Type of the hook: • PRE_PROVISIONING • POST_PROVISIONING • PRE_CONFIGUARTION • POST_CONFIGUARTION • PRE_TEARDOWN • POST_TEARDOWN Argument Type: String
`procedureName`	(Optional) Name of the procedure that the hook references. Argument Type: String
`procedurePluginKey`	(Optional) Name of the plugin procedure. Use this argument when the hook references a plugin procedure. The promoted version of the plugin is invoked by default. Argument Type: String
`procedureProjectName`	(Optional) Name of the project to which the procedure belongs. When you use a specific version of a plugin, this is the name of the plugin project. Argument Type: String
`projectName`	(Optional) Project name of the entity that owns the hook. Argument Type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument Type: String

Positional arguments

`hookName`

Response

Returns a hook for a resource template.

ec-perl

syntax: `$<object>->createHook(<hookName>, {<optionals>});`

Example

```
$ec->createHook("Config Web Server", {hooktype => PRE_CONFIGURATION, procedureNa
me => "Server Start", procedureProjectName => "Servers");
```

ectool

syntax: `ectool createHook <hookName> [optionals]`

Example

```
ectool createHook "Config Web Server" --hookType PRE_CONFIGURATION --procedureNa
me "Server Start" --procedureProjectName "Servers"
```

Back to Top

createResourceTemplate

Creates a resource template.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`resourceTemplateName`	Name for the resource template that must be unique among all resource templates. Argument Type: String
`cfgMgrParameters`	(Optional) Configuration Manager plugin parameters that are passed from the configuration-manager plugin to ElectricFlow. Argument Type: Map
`cfgMgrPluginKey`	(Optional) Configuration Manager plugin key. Argument Type: String
`cfgMgrProcedure`	(Optional) Name of the cloud-provider plugin method. Argument Type: String
`cfgMgrProjectName`	(Optional) Name of the project to which the configuration-manager plugin applies. Argument Type: String

Arguments	Descriptions
`cloudProviderParameters`	(Optional) Parameters that are passed from the cloud- provider plugin to ElectricFlow. Argument Type: Map
`cloudProviderPluginKey`	(Optional) Cloud-provider plugin key. Argument Type: String
`cloudProviderProcedure`	(Optional) Cloud-provider plugin method name. Argument Type: String
`cloudProviderConfig`	(Optional) Name of the cloud-provider plugin configuration. Argument Type: String
`cloudProviderProjectName`	(Optional) Name of the project to which the cloud-provider plugin applies. Argument Type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. This text is not interpreted by the automation platform. Argument type: String

Positional arguments

`projectName`, `resourceTemplateName`

Response

Returns a resource template.

ec-perl

syntax: `$<object>->createResourceTemplate(<projectName>, <resourceTemplateName>, {<optionals>});`

Example

```
$ec->createResourceTemplate("Default", "QE testing", {cloudProviderProjectName =
> "Deploy all"});
```

ectool

syntax: `ectool createResourceTemplate <project Name> <resourceTemplateName> [optionals]`

Example

```
ectool createResourceTemplate "Default" "QE testing" --cloudProviderProjectName
"Deploy all"
```

Back to Top

deleteEnvironmentTemplate

Deletes an environment template.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String

Positional arguments

`projectName, environmentTemplateName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteEnvironmentTemplate(<projectName>, <environmentTemplateName>);`

Example

```
$ec->deleteEnvironmentTemplate("Default", "Production");
```

ectool

syntax: `ectool deleteEnvironmentTemplate <projectName> <environmentTemplateName>`

Example

```
ectool deleteEnvironmentTemplate "Default" "Production"
```

Back to Top

deleteEnvironmentTemplateTier

Deletes an environment template tier.

You must specify the `projectName`, `environmentTemplateName`, and `environmentTemplateTierName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`environmentTemplateTierName`	Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String

Positional arguments

`projectName, environmentTemplateName, environmentTemplateTierName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <environmentTemplateTierName>);`

Example

```
$ec->deleteEnvironmentTemplateTier("Default", "Production", "Repository");
```

ectool

syntax: `ectool deleteEnvironmentTemplateTier <projectName> <environmentTemplateName> <environmentTemplateTierName>`

Example

```
ectool deleteEnvironmentTemplateTier "Default" "Production" "Repository"
```

Back to Top

deleteEnvironmentTemplateTierMap

Deletes an environment template tier map from an application.

You must specify the `projectName`, `applicationName`, `environmentProjectName`, and `environmentTemplateName` arguments.

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

Arguments	Descriptions
`applicationName`	Name of the application. Argument Type: String
`environmentProjectName`	Name for the project to which the environment template belongs. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String

Positional arguments

`projectName`, `applicationName`, `environmentProjectName`, `environmentTemplateName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteEnvironmentTemplateTierMap(<projectName>, <applicationName>, <environmentProjectName> <environmentTemplateName>);`

Example

```
$ec->deleteEnvironmentTemplateTierMap("Default", "Undeploy", "Software tools", "
Repository");
```

ectool

syntax: `ectool deleteEnvironmentTemplateTierMap <projectName> <applicationName> <environmentProjectName> <environmentTemplateName>`

Example

```
ectool deleteEnvironmentTemplateTierMap "Default" "Undeploy" "Software tools" "R
epository"
```

Back to Top

deleteEnvironmentTemplateTierMapping

Deletes a tier mapping from a environment-template tier map. A tier mapping is a mapping of an application tier to an environment template tier. A tier map has one or more mappings.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application. Argument Type: String
`environmentProjectName`	Name for the project to which the environment belongs. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`applicationTierName`	Name of the application tier. Argument Type: String

Positional arguments

`projectName`, `applicationName`, `environmentProjectName`, `environmentTemplateName`, `applicationTierName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteEnvironmentTemplateTierMapping(<projectName>, <applicationName>, <environmentProjectName> <environmentTemplateName> <applicationTierName>);`

Example:

```
$ec->deleteEnvironmentTemplateTierMapping("Default", "Undeploy", "Production", "
Repository", "Database");
```

ectool

syntax: `ectool deleteEnvironmentTemplateTierMapping <projectName> <applicationName> <environmentProjectName> <environmentTemplateName> <applicationTierName>`

Example:

```
ectool deleteEnvironmentTemplateTierMapping "Default" "Undeploy" "Production" "R
epository" "Database"
```

Back to Top

deleteHook

Deletes a hook associated with an entity.

You must specify the `hookName` argument.

Arguments	Descriptions
`hookName`	Name of the hook that must be unique among all hooks in the project. Argument Type: String
`projectName`	(Optional) Name of the project that owns the hook. Argument Type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument Type: String

Positional arguments

`hookName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteHook(<hookName>, {<optionals>});`

Example

```
$ec->deleteHook("AWS config", {resourceTemplateName => "AWS backup server"});
```

ectool

syntax: `ectool deleteHook <hookName> [optionals]`

Example

```
ectool deleteHook "AWS config" --resourceTemplateName "AWS backup server"
```

Back to Top

deleteResourceTemplate

Deletes a resource template.

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

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

Arguments	Descriptions
`resourceTemplateName`	Name for the resource template that must be unique among all resource templates. Argument Type: String

Positional arguments

`projectName`, `resourceTemplateName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteResourceTemplate(<projectName>, <resourceTemplateName>);`

Example

```
$ec->deleteResourceTemplate("Default", "QA Test");
```

ectool

syntax: `ectool deleteResourceTemplate <projectName> <resourceTemplateName>`

Example

```
ectool deleteResourceTemplate "Default" "QA Test"
```

Back to Top

getAvailableResourcesForEnvironment

Retrieves all non-dynamic resources or resource pools.

Arguments	Descriptions
`includePoolUsage`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `true` or `1`, the pool usage is also retrieved. Argument Type: Boolean
`objectTypeToReturn`	(Optional) Flag to return resources or resource pools. Valid values are `resources` or `resourcePools`. If this is set to `true` or `1`, resources or resource pools are also retrieved. Argument Type: String

Response

Returns resource or resource pool objects.

ec-perl

syntax: `$<object>->getAvailableResourcesForEnvironment({<optionals>});`

Example

```
$ec->getAvailableResourcesForEnvironment({objectTypeToReturn => "resources"});
```

ectool

syntax: `ectool getAvailableResourcesForEnvironment [optionals]`

Example

```
ectool getAvailableResourcesForEnvironment --objectTypeToReturn "resources"
```

Back to Top

getEnvironmentTemplate

Retrieves an environment template.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String

Positional arguments

`projectName, environmentTemplateName`

Response

An `environmentTemplate` element.

ec-p erl

syntax: `$<object>->getEnvironmentTemplate(<projectName>, <environmentTemplateName>);`

Example

```
$ec->getEnvironmentTemplate("Default", "QA testing");
```

ectool

syntax: `ectool getEnvironmentTemplate <projectName> <environmentTemplateName>`

Example

```
ectool getEnvironmentTemplate "Default" "QA testing"
```

Back to Top

getEnvironmentTemplateTier

Retrieves an environment tier in an environment template.

You must specify the `projectName`, `environmentTemplateName`, and `environmentTemplateTierName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`environmentTemplateTierName`	Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String

Positional arguments

`projectName, environmentTemplateName, environmentTemplateTierName`

Response

An `environmentTemplateTier` element.

ec-perl

syntax: `$<object>->getEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <environmentTemplateTierName>);`

Example

```
$ec->getEnvironmentTemplateTier("Default", "QA testing", "Repository");
```

ectool

syntax: `ectool createEnvironmentTemplateTier <projectName> <environmentTemplateName> <environmentTemplateTierName>`

Example

```
ectool createEnvironmentTemplateTier "Default" "QA testing" "Repository"
```

Back to Top

getEnvironmentTemplateTierMaps

Retrieves all the environment-template tier maps used by the specified application.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application. Argument Type: String
`applicationEntityRevisionId`	(Optional) Revision ID of the versioned object. Argument type: UUID
`orderByEnvironmentTemplateUsage`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, the most recently used environment templates is used. Argument Type: Boolean

Positional arguments

`projectName`, `applicationName`

Response

One or more `environmentTemplateTierMap` elements.

ec-perl

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

Example

```
$ec->getEnvironmentTemplateTierMaps("Default", "Undeploy");
```

ectool

syntax: `ectool getEnvironmentTemplateTierMaps <projectName> <applicationName> [optionals]`

Example

```
ectool getEnvironmentTemplateTierMaps "Default" "Undeploy"
```

Back to Top

getEnvironmentTemplateTiers

Retrieves all the environment template tiers in the specified environment template.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`includeTemplateDetails`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, the response includes the template details. Argument Type: Boolean

Positional arguments

`projectName, environmentTemplateName`

Response

One or more `environmentTemplateTier` elements.

ec-perl

syntax: `$<object>->getEnvironmentTemplateTiers(<projectName>, <environmentTemplateName>, {<optionals>});`

Example

```
$ec->getEnvironmentTemplateTiers("Default", "QA testing");
```

ectool

syntax: `ectool getEnvironmentTemplateTiers <projectName> <environmentTemplateName> [optionals]`

Example

```
ectool getEnvironmentTemplateTiers "Default" "QA testing"
```

Back to Top

getEnvironmentTemplates

Retrieves all the environment templates in the specified project.

You must specify the `projectName` argument.

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

Positional arguments

`projectName`

Response

One or more `environmentTemplate` elements.

ec-perl

syntax: `$<object>->getEnvironmentTemplates(<projectName>);`

Example

```
$ec->getEnvironmentTemplates("Default");
```

ectool

syntax: `ectool getEnvironmentTemplates <projectName>`

Example

```
ectool getEnvironmentTemplates "Default"
```

Back to Top

getHook

Retrieves a hook associated in an entity.

You must specify the `hookName`.

Arguments	Descriptions
`hookName`	Name of the hook that must be unique among all hooks in the project. Argument Type: String
`projectName`	(Optional) Name of the project to which the procedure belongs. When you use a specific version of a plugin, this is the name of the plugin project. Argument Type: String
`resourceTemplateName`	(Optional) Name of the resource template with the hook. Argument Type: String

Positional arguments

`hookName`

Response

Returns a hook for a resource template.

ec-perl

syntax: `$<object>->getHook(<hookName>, {<optionals>});`

Example

```
$ec->getHook("Config Web Server", {resourceTemplateName => "Servers"});
```

ectool

syntax: `ectool getHook <hookName> [optionals]`

Example

```
ectool getHook "Config Web Server" --resourceTemplateName "Servers"
```

Back to Top

getHooks

Retrieves all the hooks associated with an entity.

Arguments	Descriptions
`projectName`	(Optional) Name of the project to which the procedure belongs. When you use a specific version of a plugin, this is the name of the plugin project. Argument Type: String
`resourceTemplateName`	(Optional) Name of the resource template with the hook. Argument Type: String

Positional arguments

None

Response

Returns all the hooks for a resource template.

ec-perl

syntax: `$<object>->getHooks({<optionals>});`

Example

```
$ec->getHooks({projectName => "Default", resourceTemplateName => "AWS server
s"});
```

ectool

syntax: `ectool getHooks [optionals]`

Example

```
ectool getHooks --projectName "Default" --resourceTemplateName "AWS servers"
```

Back to Top

getProvisionedEnvironments

Retrieves provisioned environments.

Arguments	Descriptions
`flowRuntimeId`	(Optional) The ID of the flow state. Argument Type: UUID
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`stageName`	(Optional) Name of the stage. Argument Type: String

Positional arguments

None

Response

Returns a provisioned environment element.

ec-p erl

syntax: `$<object>->getEnvironmentTemplate({<optionals>});`

Example

```
$ec->getEnvironmentTemplate({jobId => "5da765dd-73f1-11e3-b67e-b0a420524153", st
ageName => "PROD"});
```

ectool

syntax: `ectool getEnvironmentTemplate [optionals]`

Example

```
ectool getEnvironmentTemplate --jobId "5da765dd-73f1-11e3-b67e-b0a420524153" --s
tageName "PROD"
```

Back to Top

getResourcePoolsInEnvironmentTier

Retrieves the list of resource pools in the specified environment tier.

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

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

Positional arguments

`projectName, environmentName, environmentTierName`

Response

One or more `resourcePool` elements.

ec-perl

syntax: `$<object>->getResourcePoolsInEnvironmentTier(<projectName>, <environmentName>, <environmentTierName>);`

Example

```
$cmdr->getResourcePoolsInEnvironmentTier("Default", "Production", "Web Server");
```

ectool

syntax: `ectool getResourcePoolsInEnvironmentTier <projectName> <environmentName> <environmentTierName>`

Example

```
ectool getResourcePoolsInEnvironmentTier "Default" "Production" "Web Server"
```

Back to Top

getResourceTemplate

Retrieves the specified resource template.

You must specify the `projectName` and `resourceTemplateName` argument.

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

Arguments	Descriptions
`resourceTemplateName`	Name for the resource template that must be unique among all resource templates. Argument Type: String

Positional arguments

`projectName`, `resourceTemplateName`

Response

A `resourceTemplate` element.

ec-perl

syntax: `$<object>->getResourceTemplate(<projectName>, <resourceTemplateName>);`

Example

```
$ec->getResourceTemplate("Default", "System Test");
```

ectool

syntax: `ectool getResourceTemplate <projectName> <resourceTemplateName>`

Example

```
ectool getResourceTemplate "Default" "System Test"
```

Back to Top

getResourceTemplates

Retrieves all the resource templates.

You must enter the `projectName`.

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

Positional arguments

`projectName`

Response

One or more `resourceTemplate` elements.

ec-perl

syntax: `$<object>->getResourceTemplates(<projectName>);`

Example

```
$ec->getResourceTemplates("Default");
```

ectool

syntax: `ectool getResourceTemplates <projectName>`

Example

```
ectool getResourceTemplates "Default"
```

Back to Top

getResourceTemplatesInEnvironmentTemplateTier

Retrieves all the resource templates in the specified environment template tier.

You must specify the `projectName`, `environmentTemplateName`, and `environmentTemplateTierName`arguments.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`environmentTemplateTierName`	Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String

Positional arguments

`projectName, environmentTemplateName, environmentTemplateTierName`

Response

One or more `resourceTemplate` elements.

ec-perl

syntax: `$<object>->getResourceTemplatesInEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <environmentTemplateTierName>);`

Example

```
$ec->getResourceTemplatesInEnvironmentTemplateTier("Default", "Production", "Web
Server");
```

ectool

syntax: `ectool getResourceTemplatesInEnvironmentTemplateTier <projectName> <environmentTemplateName> <environmentTemplateTierName>`

Example

```
ectool getResourceTemplatesInEnvironmentTemplateTier "Default" "Production" "Web
Server"
```

Back to Top

getResourcesInEnvironmentTemplateTier

Retrieves all the resources in the specified environment template tier.

You must specify the `projectName`, `environmentTemplateName`,and `environmentTemplateTierName`.

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`environmentTemplateTierName`	Name for the environment template tier that must be unique among all tiers for the environment template. Argument Type: String

Positional arguments

`projectName, environmentTemplateName, environmentTemplateTierName`

Response

One or more `resource` elements.

ec-perl

syntax: `$<object>->getResourcesInEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <environmentTemplateTierName>);`

Example

```
$ec->getResourcesInEnvironmentTemplateTier("Default", "QA testing", "Tomcat");
```

ectool

syntax: `ectool getResourcesInEnvironmentTemplateTier <projectName> <environmentTemplateName> <environmentTemplateTierName>`

Example

```
ectool getResourcesInEnvironmentTemplateTier "Default" "QA testing" "Tomcat"
```

Back to Top

modifyEnvironmentTemplate

Modifies an environment template.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. This text is not interpreted by the automation platform. Argument type: String
`newName`	(Optional) New name for the environment template. Argument Type: String

Positional arguments

`projectName, environmentTemplateName`

Response

Returns a modified `resourceTemplate` element.

ec-perl

syntax: `$<object>->modifyEnvironmentTemplate(<projectName>, <environmentTemplateName>, {<optionals>});`

Example

```
$ec->modifyEnvironmentTemplate("Default", "Dev Test 1");
```

ectool

syntax: `ectool modifyEnvironmentTemplate <projectName> <environmentTemplateName> [optionals]`

Example

```
ectool modifyEnvironmentTemplate "Default" "Dev Test 1"
```

Back to Top

modifyEnvironmentTemplateTier

Modifies all the environment template tiers in the specified environment template.

You must specify the `projectName`, `environmentTemplateName`, and `environmentTemplateTierName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`environmentTemplateTierName`	Name for the environment template tier that must be unique among all tiers for the environment template. Argument Type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. This text is not interpreted by the automation platform. Argument type: String
`newName`	(Optional) New name for the environment template tier. Argument Type: String
`resourceCount`	(Optional) Number of resources to be spun from the resource template. Argument Type: Integer
`resourceNames`	(Optional) List of resources to add to the environment tier. Argument type: Collection
`resourceTemplateName`	(Optional) Name for the resource template that must be unique among all resource templates. Argument type: String
`resourceTemplateProjectName`	(Optional) Name of the project to which the resource template belongs. Argument Type: String

Positional arguments

`projectName, environmentTemplateName, environmentTemplateTierName`

Response

Returns a modified `environmentTemplateTier` element.

ec-perl

syntax: `$<object>->modifyEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <environmentTemplateTierName>,{<optionals>});`

Example

```
$ec->modifyEnvironmentTemplateTier("Default", "QA test", "Database");
```

ectool

syntax: `ectool modifyEnvironmentTemplateTier <projectName> <environmentTemplateName> <environmentTemplateTierName> [optionals]`

Example

```
ectool modifyEnvironmentTemplateTier "Default" "QA test" "Database"
```

Back to Top

modifyEnvironmentTemplateTierMap

Modifies an existing environment template tier map.

You must specify the `projectName`, `applicationName`, `environmentProjectName`, and `environmentTemplateName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application. Argument Type: String
`environmentProjectName`	Name for the project to which the environment belongs. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`applicationEntityRevisionId`	(Optional) Revision ID of the versioned object. Argument Type: UUID
`tierMapName`	(Optional) Name of the tier map associated with the environment template. If you do not specify an tier map, ElectricFlow uses a tier map with a hyphenated-application-and-environment name. Argument type: String

Arguments	Descriptions
`tierMappings`	(Optional) List of mappings between the application tiers and the environment template tiers. Argument Type: Map

Positional arguments

`projectName`, `applicationName`, `environmentProjectName`, `environmentTemplateName`

Response

Returns a modified `environmentTemplateTier` element.

ec-perl

syntax: `$<object>->modifyEnvironmentTemplateTierMap(<projectName>, <applicationName>, <environmentProjectName>, <environmentTemplateName>, {<optionals>});`

Example

```
$ec->modifyEnvironmentTemplateTierMap("Default", "Undeploy", "Utilities", "Serve
rs");
```

ectool

syntax: `ectool modifyEnvironmentTemplateTierMap <projectName> <applicationName> <environmentProjectName> <environmentTemplateName> [optionals]`

Example

```
ectool modifyEnvironmentTemplateTierMap "Default" "Undeploy" "Utilities" "Server
s"
```

Back to Top

modifyEnvTemplTierResourceTemplMapping

Modifies the resource count in an environment template tier.

You must specify the `resourceCount`, `projectName`, `environmentTemplateName`, `resourceTemplateName`, and `environmentTemplateTierName`.

Arguments	Descriptions
`resourceCount`	Number of resources to provision from the specified resource template. Argument Type: Integer
`projectName`	Name for the project that must be unique among all projects. Argument Type: String

Arguments	Descriptions
`environmentTemplateName`	Name of the environment template. Argument Type: String
`resourceTemplateName`	Name of the resource template that must be unique among all resource templates. Argument Type: String
`environmentTemplateTierName`	Name for the environment template tier that must be unique among all tiers for the environment template. Argument Type: String
`resourceTemplateProjectName`	(Optional) Name for the project to which the resource template belongs. Argument Type: String

Positional arguments

`resourceCount`, `projectName`, `environmentTemplateName`, `resourceTemplateName`, `environmentTemplateTierName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->modifyEnvTempTierResourceTemplMapping(<resourceCount>, <projectName>, <environmentTemplateName>, <resourceTemplateName>, <environmentTemplateTierName>, {<optionals>});`

Example

```
$ec->modifyEnvTempTierResourceTemplMapping(5, "Servers", "Default", Dev1", "Data
base", {resourceTemplateProjectName =>"Utilities" });
```

ectool

syntax: `ectool modifyEnvTempTierResourceTemplMapping <resourceCount> <resourceTemplateName> <projectName> <environmentTemplateName> <environmentTemplateTierName>`

Example

```
ectool modifyEnvTempTierResourceTemplMapping 5 "Servers" "default" Dev1" "Databa
se" --resourceTemplateProjectName "Utilities"
```

Back to Top

modifyHook

Modifies an existing hook in a resource template.

You must specify the `hookName`.

Arguments	Descriptions
`hookName`	Name of the hook that must be unique among all hooks in the project. Argument Type: String
`broadcast`	(Optional) Broadcast flag Use this flag to broadcast the hook name in the project. The `broadcast` value = <*Boolean flag* -`0|1|true|false`>. Defaults to `true` or `1`. Argument type: Boolean
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. This text is not interpreted by the automation platform. Argument type: String
`hookParameters`	(Optional) Parameters that are passed to the procedure. Argument type: Map
`hookType`	(Optional)Type of the hook: • PRE_PROVISIONING • POST_PROVISIONING • PRE_CONFIGUARTION • POST_CONFIGUARTION • PRE_TEARDOWN • POST_TEARDOWN Argument Type: String
`newName`	New name for the hook.
`procedureName`	Name of the procedure that the hook references. Argument Type: String
`procedurePluginKey`	(Optional) Name of the plugin procedure. Use this argument when the hook references a plugin procedure. The promoted version of the plugin is invoked by default. Argument Type: String

Arguments	Descriptions
`procedureProjectName`	(Optional) Name of the project to which the procedure belongs. When you use a specific version of a plugin, this is the name of the plugin project. Argument Type: String
`projectName`	(Optional) Project name of the entity that owns the hook. Argument Type: String
`resourceTemplateName`	(Optional) Name of the resource template with the hook. Argument Type: String

Positional arguments

`hookName`

Response

None or a status OK message.

ec-perl

syntax:`$<object>->modifyHook(<hookName>, {<optionals>});`

Example

```
$ec->modifyHook("config", {newName => "prod_config"});
```

ectool

syntax: `ectool modifyHook <hookName> [optionals]`

Example

```
ectool modifyHook "config" --newName  "prod_config"
```

Back to Top

modifyResourceTemplate

Modifies the specified resource template.

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

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

Arguments	Descriptions
`resourceTemplateName`	Name for the resource template that must be unique among all resource templates. Argument Type: String
`cfgMgrParameters`	(Optional) Parameters that are passed from the configuration-manager plugin to ElectricFlow. Argument type: Map
`cfgMgrPluginKey`	(Optional) Name of the configuration-manager plugin. Argument Type: String
`cfgMgrProcedure`	(Optional) Name of the configuration-manager plugin method. Argument Type: String
`cfgMgrProjectName`	(Optional) Name of the project to which the configuration-manager plugin applies. Argument Type: String
`cloudProviderParameters`	(Optional) Parameters that are passed from the cloud- provider plugin to ElectricFlow. Argument Type: Map
`cloudProviderPluginKey`	(Optional) Name of the cloud-provider plugin. Parameters that are passed from the cloud-provider plugin to ElectricFlow. Argument Type: String
`cloudProviderProcedure`	Name of the cloud-provider plugin method. Argument Type: String
`cloudProviderProjectName`	(Optional) Name of the project to which the cloud-provider plugin applies. Argument Type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. This text is not interpreted by the automation platform. Argument type: String
`newName`	New name for the resource template. Argument type: String

Positional arguments

`projectName`, `resourceTemplateName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->modifyResourceTemplate(<projectName>, <resourceTemplateName>, {<optionals>});`

Example

```
$ec->modifyResourceTemplate("Default", "System Test", {newName => "System Test 1
"});
```

ectool

syntax: `ectool modifyResourceTemplate <projectName> <resourceTemplateName> [optionals]`

Example

```
ectool modifyResourceTemplate "Default" "System Test" --newName "System Test 1"
```

Back to Top

provisionEnvironment

Provisions an environment.

You must specify the `projectName`, `environmentName`, and `environmentTemplateName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects Argument Type: String
`environmentName`	Name of the environment. Argument Type: String
`environmentTemplateName`	Name for the environment template that must be unique among all environment templates. Argument Type: String
`environmentTemplateProjectName`	(Optional) Name of the project containing specified environment template. If this argument is not specified, the default is the environment project name. Argument Type: String

Arguments	Descriptions
`keepOnError`	(Optional) The `keepOnError` error flag value is *<Boolean flag -* `0|1|true|false`>. Set this flag to `1` or `true` to keep the environment when an error occurs .The default is `0` or `false`. Argument type: Boolean
`tierResourceCounts`	(Optional) Resource count for each environment template tier. Argument type: Map

Positional arguments

`projectName`, `environmentName`, `environmentTemplateName`

Response

Returns a `jobId`. You need to wait for the job to be completed before using the resources.

ec-perl

syntax:`$<object>->provisionEnvironment(<projectName>, <environmentName>, <environmentTemplateName>, {<optionals>});`

Example

```
$ec->provisionEnvironment("Default", "Main servers", "Build Server", {keepOnErro
r => 1});
```

ectool

syntax: `ectool provisionEnvironment <projectName> <environmentName> <environmentTemplateName> [optionals]`

Example

```
ectool provisionEnvironment "Default" "Main servers" "Build Server" --keepOnErro
r 1
```

Back to Top

provisionResourcePool

Provisions a resource pool.

You must specify the `resourceCount`, `resourcePoolName`, `projectName`, and `resourceTemplateName`.

Arguments	Descriptions
`resourceCount`	Number of resources to provision. Argument Type: String

Arguments	Descriptions
`resourcePoolName`	Name of the resource pool. Argument Type: String
`projectName`	Name of the project that must be unique among all projects Argument Type: String
`resourceTemplateName`	Name for the resource template that must be unique among all resource templates. Argument Type: String
`keepOnError`	(Optional) The `keepOnError` error flag value is <*Boolean flag* - `0|1|true|false`>. Set this flag to `1` or `true` to keep the environment when an error occurs .The default is `0` or `false`. Argument type: Boolean

Positional arguments

`resourceCount`, `resourcePoolName`, `projectName`, `resourceTemplateName`

Response

Returns a `jobId`. You need to wait for the job to be completed before using the resources.

ec-perl

syntax: `$<object>->provisionResourcePool(<resourceCount>, <resourcePoolName>, <projectName>, <resourceTemplateName>,{<optionals>});`

Example

```
$ec->provisionResourcePool("12", "QE build", "Default",  "Servers", {keepOnError
=> 1});
```

ectool

syntax: `ectool provisionResourcePool <resourceCount> <resourcePoolName> <projectName> <resourceTemplateName> [optionals]`

Example

```
ectool provisionResourcePool "12" "QE build" "Default" "Servers" --keepOnError 1
```

Back to Top

removeResourceFromEnvironmentTemplateTier

Removes a resource from an environment template tier.

You must specify the `resourceName`, `projectName`, `environmentTemplateName`, and `environmentTemplateTierName`

Arguments	Descriptions
`resourceName`	Name of the resource that must be unique among all resources. Argument Type: String
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`environmentTemplateTierName`	Name for the environment template tier that must be unique among all tiers for the environment template. Argument Type: String

Positional arguments

`resourceName, projectName, environmentTemplateName, environmentTemplateTierName`

Response

Removes the specified resource from the environment template tier.

ec-perl

syntax: `$<object>->removeResourceFromEnvironmentTemplateTier(<resourceName>, <projectName>, <environmentTemplateName>, <environmentTemplateTierName>);`

Example

```
$ec->removeResourceFromEnvironmentTemplateTier("Web Server", "Default", "QA la
b", "Tomcat");
```

ectool

syntax:`ectool removeResourceFromEnvironmentTemplateTier <resourceName> <projectName> <environmentTemplateName> <environmentTemplateTierName>`

Example

```
ectool removeResourceFromEnvironmentTemplateTier "Web Server" "Default" "QA lab"
"Tomcat"
```

Back to Top

removeResourcePoolFromEnvironmentTier

Removes a resource pool from the specified environment tier.

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

Arguments	Descriptions
`resourcePoolName`	Name of the resource pool that must be unique among all resource pools. Argument Type: String
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentName`	Name of the environment that must be unique among all environments. Argument Type: String
`environmentTierName`	Name of the environment tier that must be unique among all tiers for the environment. Argument Type: String

Positional arguments

`resourcePoolName, projectName, environmentName, environmentTierName`

Response

None or status OK message.

ec-perl

syntax:`$<object>->removeResourcePoolFromEnvironmentTier (<resourcePoolName>, <projectName>, <environmentName>, <environmentTierName>);`

Example

```
$ec->removeResourcePoolFromEnvironmentTier("pool1", "Default", "Production", "Web S
erver");
```

ectool

syntax:`ectool removeResourcePoolFromEnvironmentTier <resourcePoolName> <projectName> <environmentName> <environmentTierName>`

Example

```
ectool removeResourcePoolFromEnvironmentTier "pool1" "Default" "Production" "Web Se
rver"
```

Back to Top

removeResourceTemplateFromEnvironmentTemplateTier

Removes a resource template from the specified environment template tier.

You must specify the `resourceTemplateName`, `projectName`, `environmentTemplateName`, and `environmentTemplateTierName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`environmentTemplateName`	Name of the environment template. Argument Type: String
`resourceTemplateName`	Name of the resource template that must be unique among all resource templates. Argument Type: String
`environmentTemplateTierName`	Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String
`resourceTemplateProjectName`	(Optional) Name for the project to which the resource template belongs. Argument Type: String

Positional arguments

`projectName, environmentTemplateName, resourceTemplateName, environmentTemplateTierName`

Response

None or a status OK message.

ec-perl

syntax:`$<object>->removeResourceTemplateFromEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <resourceTemplateName>, <environmentTemplateTierName>, {<optionals>});`

Example

```
$ec->removeResourceTemplateFromEnvironmentTemplateTier("Default", "QA lab", "Tes
ter 1", "App Server", {resourceTemplateProjectName => "QA test"});
```

ectool

syntax:`ectool removeResourceTemplateFromEnvironmentTemplateTier <resourceTemplateName> <projectName> <environmentTemplateName> <environmentTemplateTierName> [optionals]`

Example

```
ectool removeResourceTemplateFromEnvironmentTemplateTier "Default", "QA lab", "T
ester 1", "App Server" --resourceTemplateProjectName "QA test"
```

Back to Top

tearDown

Removes (decommissions) dynamic environments that are no longer needed.

Arguments	Descriptions
`environmentName`	(Optional) Name of the environment. Argument Type: String
`pollInterval`	(Optional) This argument requires setting a value in *seconds* to determine how often ectool queries the ElectricFlow 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. If this option is not specified, `runProcedure` returns immediately. If it is specified, `runProcedure` waits until the job completes. Argument type: Integer
`projectName`	(Optional) Name of the project that must be unique among all projects. Argument Type: String
`resourceName`	(Optional) Name of the resource. Argument Type: String
`resourcePoolName`	(Optional) Name of the resource pool. Argument Type: String
`timeout`	(Optional) 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. Argument type: Integer

Positional arguments

None

Response

Returns a `jobId`.

If the `--pollInterval` option is provided, ElectricFlow wait until the job completes up to a maximum of `--timeout` seconds (if also provided).

ec-perl

To tear down (decommission) an environment:

syntax:`$<object>->tearDown ({<optionals>});`

Example

```
$ec->tearDown ({environmentName => "Server backup", projectName => "Default"});
```

ectool

To tear down (decommission) an environment:

syntax: `ectool tearDown [optionals]`

Example

```
ectool tearDown --environmentName "Server backup" --projectName "Default"
```

Back to Top

API Commands - Email Configuration Management

createEmailConfig

Creates a new email configuration.

You must specify `configName`.

Arguments	Descriptions
`configName`	The name of your email configuration. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`mailFrom`	(Optional) The email address used as the email sender address for notifications. Argument type: String

Arguments	Descriptions
`mailHost`	(Optional) The name of the email server host. Argument type: String
`mailPort`	(Optional) 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. Argument type: String
`mailProtocol`	(Optional) This is either SSMTP or SMTP (not case-sensitive). The default is SMTP.
`mailUser`	(Optional) This can be an individual or a generic name like "ElectricFlow" - name of the email user on whose behalf ElectricFlow sends email notifications. Argument type: String
`mailUserPassword`	(Optional) Password for the email user who is sending notifications. Argument type: String

Positional arguments

`configName`

Response

Returns an `emailConfig` element.

ec-perl

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

Example

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

ectool

syntax: `ectool createEmailConfig <configName> ...`

Example

```
ectool createEmailConfig EmailConfig_test --mailHost ectest-sol2
  --mailFrom ElectricFlow@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. Argument Type: String

Positional arguments

`configName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteEmailConfig("All users");
```

ectool

syntax: `ectool deleteEmailConfig <configName>`

Example

```
ectool deleteEmailConfig "All users"
```

Back to Top

getEmailConfig

Retrieves an email configuration by name.

You must specify a `configName`.

Arguments	Descriptions
`configName`	The name of the email configuration. Argument Type: String

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("Config_test");
```

ectool

syntax: `ectool getEmailConfig <configName>`

Example

```
ectool getEmailConfig "Config_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 ElectricFlow 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. Argument Type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. Argument Type: String
`mailFrom`	(Optional) The email address used as the email "sender" address for notifications. Argument Type: String
`mailHost`	(Optional) The name of the email server host. Argument Type: String
`mailPort`	(Optional) 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. Argument Type: Integer
`mailProtocol`	(Optional) This is either SSMTP or SMTP (not case-sensitive). The default is SMTP. Argument Type: String
`mailUser`	(Optional) The name of the email user, which can be an individual or a generic name such as "ElectricFlow". Argument Type: String

Arguments	Descriptions
`mailUserPassword`	(Optional) The password for the email user. Argument Type: String
`newName`	(Optional) New name of the email configuration. Argument Type: String

Positional arguments

`configName`

Response

Returns a modified `emailConfig` element.

ec-perl

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

Example

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

ectool

syntax: `ectool modifyEmailConfig <configName> [optionals]`

Example

```
ectool modifyEmailConfig "QA test" --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, such as a job, job step, project, procedure, application process or process step, or a workflow.

You must specify a `notifierName` and object locators for a job, job step, procedure, or procedure step.

Arguments	Descriptions
`notifierName`	The name of the email notifier. Argument type: String
`applicationName`	(Optional) The name of the application that is related to the target email container. The email notifier is attached to an application process or process step. Argument type: String
`componentName`	(Optional) The name of the component that is related to the target email container. The email notifier is attached to the component. Argument type: String
`condition`	(Optional) 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. Argument type: String
`configName`	(Optional) If specified, this argument must be the name of an `emailConfig` object. If it is not specified, the default is the name of the first `emailConfig` object defined for the ElectricFlow server (`emailConfig` objects are "ordered" ElectricFlow entities). **Note:** When using this argument, you must also include the `formattingTemplate` or the `formattingTemplateFile` argument . Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String

Arguments	Descriptions
`destinations`	(Optional) A mandatory argument for a `create` operation. A space-separated list of valid email addresses, email aliases, or ElectricFlow user names, or a string subject to property expansion that expands into such a list. Argument type: String
`environmentNames`	(Optional) The environment names. Argument type: Collection
`eventType`	(Optional) Use one of these values: <`onStart|onCompletion`>. • `onStart`–Triggers an event when the job or job step begins. • `onCompletion`–Triggers an event when the job ends, regardless of the results. The default is `onCompletion`. Argument type: EventType
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`formattingTemplate`	(Optional) A template for formatting email messages when an event [notification] is triggered by the `emailNotifier`. Make sure the content is formatted correctly, such as no illegal characters or spacing. Argument type: String
`formattingTemplateFile`	(Optional) **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.
`gateType`	(Optional) The type of the gate. Argument type: GateType
`groupNames`	(Optional) A list of groups that receive the notification. Argument type: Collection

Arguments	Descriptions
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notificationType`	(Optional) The notification type that is stored to the `ec_ notificationType` property. Argument type: NotificationType
`pipelineName`	(Optional) The name of the pipeline when a credential attached to a stage task.
`procedureName`	(Optional) The name of the procedure. When using this argument, you must also enter the `projectName`. Argument type: String
`processName`	(Optional) The name of the process that contains of the email notifier. Argument type: String
`processStepName`	(Optional) The name of the process step that contains of the email notifier. Argument type: String
`projectName`	(Optional) The name of the project. When using this argument, you must also enter the `procedureName`. Argument type: String
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String

Arguments	Descriptions
`stepName`	(Optional) The name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`userNames`	(Optional) The names of the users who receive the notification. Argument type: Collection
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String

Positional arguments

`notifierName`

Response

Returns an `emailNotifier` element.

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> [optionals]`

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

createEventSubscription

Creates a list of event subscriptions.

You must specify a `notifierName` .

Arguments	Descriptions
`notifierName`	The name of the email notifier. Argument type: String
`applicationName`	(Optional) The name of the application that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`componentName`	The name of the component that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`gateType`	(Optional) The type of the gate. Argument type: GateType
`groupNames`	A list of groups that receive the notification. Argument type: Collection
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID

Arguments	Descriptions
`jobStepId`	The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`procedureName`	The name of the procedure. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	The name of the process. Argument type: String
`processStepName`	The name of the process step. Argument type: String
`projectName`	The name of the project. When using this argument, you must also enter `procedureName`. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`stateName`	The name of the state. Argument type: String
`stepName`	The name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`userNames`	The names of the users who receives the notification. Argument type: Collection
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`workflowName`	The name of the workflow. Argument type: String

Positional arguments

`notifierName`

Response

Returns an `eventSubscription` element.

ec-perl

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

Example

```
$cmdr->createEventSubscription ("testNotifier", {applicationName => "Pass or Fai
l"});
```

ectool

syntax: `ectool createEventSubscription <notifierName> [optionals]`

Example

```
ectool createEventSubscription testNotifier --applicationName "Pass or Fail"
```

Back to Top

deleteEmailNotifier

Deletes an email notifier from an object.

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

Arguments	Descriptions
`notifierName`	The name of the email notifier that you want to delete. Argument type: String
`Locator arguments:`	
`applicationName`	The name of the application that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`componentName`	The name of the component that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String

Arguments	Descriptions
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`gateType`	(Optional) The type of the gate. Argument type: GateType
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`procedureName`	The name of the procedure. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	Name of the process. Argument type: String
`processStepName`	Name of the process step. Argument type: String
`projectName`	Name of the project. Argument type: String
`stageName`	The name of the stage definition. Argument type: String
`stateDefinitionName`	Name of the state definition. Argument type: String
`stateName`	Name of the state. Argument type: String
`stepName`	Name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String

Arguments	Descriptions
`taskName`	(Optional) The name of the task. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`workflowName`	The name of the workflow. Argument type: String

Positional arguments

`notifierName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteEmailNotifier <notifierName> [optionals]`

Example

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

Back to Top

deleteEventSubscription

Deletes a list of event subscriptions.

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

Arguments	Descriptions
`notifierName`	The name of the email notifier. Argument type: String

Arguments	Descriptions
`applicationName`	(Optional) The name of the application that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`componentName`	(Optional) The name of the component that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`gateType`	(Optional) The type of the gate. Argument type: GateType
`groupNames`	(Optional) A list of groups that receive the notification. Argument type: Collection
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`procedureName`	(Optional) The name of the procedure. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) Name of the process. Argument type: String

Arguments	Descriptions
`processStepName`	(Optional) Name of the process step. Argument type: String
`projectName`	(Optional) The name of the project. When using this argument, you must also enter `procedureName`. Argument type: String
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) Name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`taskName`	(Optional) The name of the task. Argument type: String
`userNames`	(Optional) A list of names of the users who receives the notification. Argument type: Collection
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String

Positional arguments

`notifierName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteEventSubscription (mySubscription, {applicationName => myAppEvent});
```

ectool

syntax: `ectool deleteEventSubscription<notifierName> [optionals]`

Example

```
ectool deleteEventSubscription mySubscription --applicationName myAppEvent
```

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
`notifierName`	The name of the email notifier. Argument type: String
`applicationName`	(Optional) The name of the application that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`componentName`	(Optional) The name of the component that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`gateType`	(Optional) Type of the gate. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID

Arguments	Descriptions
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`procedureName`	(Optional) The name of the procedure. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) Name of the process. Argument type: String
`processStepName`	(Optional) Name of the process step. Argument type: String
`projectName`	(Optional) The name of the project. When using this argument, you must also enter `procedureName`. Argument type: String
`stageName`	(Optional) The name of the pipeline stage. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`stateName`	The name of the state. Argument type: String
`stepName`	Name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`workflowName`	The name of the workflow. Argument type: String

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> [optionals]`

Example

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

Back to Top

getEmailNotifiers

Retrieves all email notifiers defined for the specified property sheet.

You must specify one or more object locators.

Arguments	Descriptions
applicationName	The name of the application that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
componentName	The name of the component that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
flowName	(Optional) The name of the flow. Argument type: String
flowStateName	(Optional) The name of the flow state. Argument type: String
gateType	(Optional) Type of the gate. Argument type: String

Arguments	Descriptions
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`procedureName`	(Optional) The name of the procedure. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) The name of the process. Argument type: String
`processStepName`	(Optional) The name of the process step. Argument type: String
`projectName`	(Optional) The name of the project . When using this argument, you must also enter `procedureName`. Argument type: String
`stageName`	(Optional) The name of the pipeline stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String

Arguments	Descriptions
`workflowName`	The name of the workflow. Argument type: String

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 [optionals]`

Example

```
ectool getEmailNotifiers --projectName "Test" --procedureName "Build"
```

Back to Top

getEventSubscription

Retrieves an event subscription for the specified user or group.

You must specify a `notifierName`.

Arguments	Descriptions
`notifierName`	The name of the email notifier. Argument type: String
`applicationName`	(Optional) The name of the application that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`componentName`	(Optional) The name of the component that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String

Arguments	Descriptions
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`gateType`	(Optional) Type of the gate. Argument type: String
`groupName`	The name of the group that receives the notification. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`procedureName`	(Optional) The name of the procedure. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	Name of the process. Argument type: String
`processStepName`	Name of the process step. Argument type: String
`projectName`	The name of the project . When using this argument, you must also enter `procedureName`. Argument type: String
`stageName`	(Optional) The name of the pipeline stage. Argument type: String

Arguments	Descriptions
`stateDefinitionName`	The name of the state definition. Argument type: String
`stateName`	The name of the state. Argument type: String
`stepName`	Name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`taskName`	(Optional) The name of the task. Argument type: String
`userName`	The names of the users who receives the notification. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`workflowName`	The name of the workflow. Argument type: String

Positional arguments

`notifierName`

Response

Returns an event subscription for a user or group.

ec-perl

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

Example

```
$cmdr->getEventSubscription("Error", {groupName => "QA"});
```

ectool

syntax: `ectool getEventSubscription <notifierName> [optionals]`

Example

```
ectool getEventSubscription "Error" --groupName "QA"
```

Back to Top

getEventSubscriptions

Retrieves a list event subscriptions for a specified event.

You must specify a `notifierName`.

Arguments	Descriptions
`notifierName`	The name of the email notifier. Argument type: String
`applicationName`	(Optional) The name of the application that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`componentName`	(Optional) The name of the component that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`gateType`	(Optional) Type of the gate. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: String
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String

Arguments	Descriptions
`procedureName`	(Optional) The name of the procedure. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) Name of the process. Argument type: String
`processStepName`	(Optional) Name of the process step. Argument type: String
`projectName`	(Optional) The name of the project . When using this argument, you must also enter `procedureName`. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) Name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String

Positional arguments

`notifierName`

Response

Returns an event subscription for a specific event.

ec-perl

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

Example

```
$cmdr->getEventSubscriptions("Error", {applicationName => "Pet Store"});
```

ectool

syntax: `ectool getEventSubscriptions <notifierName> [optionals]`

Example

```
ectool getEventSubscriptions "Error" --applicationName "Pet Store"
```

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
`notifierName`	The name of the email notifier. Argument type: String
`applicationName`	(Optional) The name of the application that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`componentName`	(Optional) The name of the component that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`condition`	(Optional) 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. Argument type: String

Arguments	Descriptions
`configName`	(Optional) If specified, this argument must be the name of an `emailConfig` object. If it is not specified, the default is the name of the first `emailConfig` object defined for the ElectricFlow server (`emailConfig` objects are "ordered" ElectricFlow entities). **Note:** When using this argument, you must also include the `formattingTemplate` or the `formattingTemplateFile` argument . Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. Argument type: String
`destinations`	(Optional) A space-separated list of valid email addresses, email aliases, or ElectricFlow user names, or a string subject to property expansion that expands into such a list. **Note:** This argument is mandatory for the "create" operation. Argument type: String
`environmentNames`	(Optional) The names of the environments. Argument type: Collection
`eventType`	(Optional) Use one of these values: <`onStart|onCompletion`>. • `onStart`–Triggers an event when the job or job step begins. • `onCompletion`–Triggers an event when the job ends, regardless of the results. The default is `onCompletion`. Argument type: EventType
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String

Arguments	Descriptions
`formattingTemplate`	(Optional) A template for formatting email messages when an event [notification] is triggered by the `emailNotifier`. Make sure the content is formatted correctly, such as no illegal characters or spacing. Argument type: String
`formattingTemplateFile`	(Optional) **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.
`gateType`	(Optional) Type of the gate. Argument type: String
`groupNames`	(Optional) The list of the groups that receives the notification. Argument type: Collection
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`newName`	(Optional) New name of the email notifier. Argument type: String
`notificationType`	(Optional) The notification type that is stored to the `ec_notificationType` property. Argument type: NotificationType
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`procedureName`	(Optional) The name of the procedure. When using this argument, you must also enter the `projectName`. Argument type: String

Arguments	Descriptions
processName	(Optional) The name of the process that contains of the email notifier. Argument type: String
processStepName	(Optional) The name of the process step that contains of the email notifier. Argument type: String
projectName	(Optional) The name of the project. When using this argument, you must also enter the procedureName. Argument type: String
stageName	(Optional) The name of the stage definition. Argument type: String
stateDefinitionName	(Optional) The name of the state definition. Argument type: String
stateName	(Optional) The name of the state. Argument type: String
stepName	(Optional) Name of the step. When using this argument, you must also enter projectName and procedureName. Argument type: String
taskName	(Optional) The name of the task. Argument type: String
userNames	(Optional) The names of the users who receives the notification. Argument type: Collection
workflowDefinitionName	(Optional) The name of the workflow definition. Argument type: String
workflowName	(Optional) The name of the workflow. Argument type: String

Positional arguments

notifierName

Response

Returns a modified emailNotifier element.

ec-perl

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

Example

```
$cmdr->modifyEmailNotifier("testNotifier",
        {eventType => "onCompletion",
       projectName => "Project_test",
     procedureName => "Procedure_test",});
```

ectool

syntax: `ectool modifyEmailNotifier <notifierName> [optionals]`

Example

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

Back to Top

modifyEventSubscription

Modifies a list of event subscriptions.

You must specify a `notifierName`.

Arguments	Descriptions
`notifierName`	The name of the email notifier. Argument type: String
`applicationName`	(Optional) The name of the application that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`componentName`	(Optional) The name of the component that is related to the target email container. The email notifier is attached to a process or process step. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`gateType`	(Optional) Type of the gate. Argument type: String

Arguments	Descriptions
`groupNames`	(Optional) The list of the groups that receives the notification. Argument type: Collection
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`procedureName`	(Optional) The name of the procedure. When using this argument, you must also enter the `projectName`. Argument type: String
`processName`	(Optional) The name of the process that contains of the email notifier. Argument type: String
`processStepName`	(Optional) The name of the process step that contains of the email notifier. Argument type: String
`projectName`	(Optional) The name of the project. When using this argument, you must also enter the `procedureName`. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String

Arguments	Descriptions
`stepName`	(Optional) The name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`taskName`	(Optional) The name of the task. Argument type: String
`userNames`	(Optional) The names of the users who receives the notification. Argument type: Collection
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String

Positional arguments

`notifierName`

Response

Returns a modified event subscription for a user or group.

ec-perl

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

Example

```
$cmdr->modifyEventSubscription("Error", {componentName => "Config files"});
```

ectool

syntax: `ectool modifyEventSubscription <notifierName> [optionals]`

Example

```
ectool getEventSubscription "Error" --componentName "Config files"
```

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
`attachment`	One or more client-side files to send as attachments. The filename extension is examined to determine the content type. You can enter this argument more than once to specify multiple attachments. Argument type: Collection
`bcc`	A "Bcc" recipient for the email message. The recipient can be a user name, group name or complete email address. You can enter this argument more than once to specify multiple recipients. Argument type: Collection
`cc`	A "Cc" recipient for the email message. The recipient can be a user name, group name or complete email address. You can enter this argument more than once to specify multiple recipients. Argument type: Collection
`configName`	The name of the email configuration. If no configuration is specified, the configuration named "default" is used. **Note:** The user must have *execute* permission on the configuration. Argument type: String
`header`	An RFC822 email header line (for example: "reply-to: user@host.com"). This option can be specified multiple times. Argument type: Collection
`html`	The body of a simple HTML message. Argument type: String
`htmlFile`	Reads the specified client-side file and uses it as the body of a simple HTML message.

Arguments	Descriptions
`inline`	Inline attachments in this format: `<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`* You can enter this argument more than once to add multiple attachments. Argument type: Collection
`mailFrom`	The email address used in the "From" header to use when sending ElectricFlow notification email. When the email configuration is specified, this value overrides the value in the configuration. Argument type: String
`mailHost`	The name of the email server host to use when the `configName` is not specified. When the email configuration is specified, this value overrides the value in the configuration. Argument type: String
`mailPort`	The mail server port to use when the `configName` is not specified. When the email configuration is specified, this value overrides the value in the configuration. Argument type: Integer
`mailProtocol`	The name of the mail protocol that must be SMTP or SMTPS. When the email configuration is specified, this value overrides the value in the configuration. Argument type: String
`mailUser`	The name of the email user for whom ElectricFlow sends email notifications. ElectricFlow also uses it when authenticating to the mail server. When the email configuration is specified, this value overrides the value in the configuration. Argument type: String
`mailUserPassword`	The password that ElectricFlow uses when when authenticating to the mail server. When the email configuration is specified, this value overrides the value in the configuration. Argument type: String

Arguments	Descriptions
multipartMode	The multipart message mode: `<none|mixed|related|mixedRelated>` • `none`– Non-multipart message • `mixed`–Single-root multipart element of type "mixed". Texts, inline elements, and attachments will be added to this root element. • `related`– Multipart message with a single root multipart element of "related" type. Texts, inline elements, and attachments will be added to this root element. It works on most mail clients, except Lotus Notes. • `mixedRelated`–Multipart "mixed" element with a nested multipart "related" element. Texts and inline elements will be added to the nested "related' element, while attachments will be added to the "mixed' root element. It works on most mail clients other than Mac Mail and some situations on Outlook. If you experience problems, use the "related" value. **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. Argument type: MultipartMode
raw	A raw RFC 822 email message including headers to use as the basis for the email message. Additional options can be applied to this message. Argument type: String
rawFile	Reads the specified client-side file and uses it as the entire mail message, including headers.
subject	The subject of the email message. Argument type: String
text	The body of a simple text message. Argument type: String
textFile	Reads the specified client-side file and uses it as the body of a simple text message.
to	A "To" recipient for the email message. The recipient can be a user, group name or complete email address. You can enter this argument more than once to specify multiple recipients. Argument type: Collection

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

createEnvironment

Creates a new environment.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

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

Argument Type: String

Optional Arguments

`applicationName`

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

Argument Type: String

`applicationProjectName`

Description: Name of the project that contains the application.

Argument Type: String

`description`

Description: Comment text describing this object; not interpreted by ElectricFlow.

Argument Type: String

`environmentEnabled`

Description: True to enable rolling deploy on this environment.

Argument Type: Boolean

`reservationRequired`

Description: <*Boolean flag* -`0|1|true|false`>
Set this flag to "true" to allow deployments to run using only using environment reservations.

Argument Type: Boolean

`rollingDeployEnabled`

Description: <*Boolean flag* -`0|1|true|false`>
Set this flag to "true" or "1" to enable the environment.

Argument Type: Boolean

`rollingDeployType`

Description: The type of rolling deploy supported by the environment.

Argument Type: RollingDeployType

Response

Returns an environment element.

ec-perl

Syntax:

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

Example:

```
$ec->createEnvironment("Default", "Web Server", {environmentEnabled => true, des
cription => "Web Services", rollingDeployEnabled => true});
```

ectool

Syntax:

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

Example:

```
ectool createEnvironment "Default" "Web Server" --environmentEnabled true --desc
ription "Web Server" --rollingDeployEnabled true
```

Back to Top

createEnvironmentInventoryItem

Creates a new environment inventory item.

Required Arguments

`projectName`

Description: Name for the project that 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

Optional Arguments

`applicationTierName`

Description: Name of the application tier.

Argument Type: String

`artifactName`

Description: Artifact name for the inventory item.

Argument Type: String

`artifactSource`

Description: Source of the artifact.

Argument Type: String

`artifactUrl`

Description: URL of the artifact.

Argument Type: String

`artifactVersion`

Description: Artifact version for the inventory item.

Argument Type: String

`description`

Description: Comment text describing this object; not interpreted by ElectricFlow.

Argument Type: String

`environmentProjectName`

Description: Name of the project to which the environment or environment template belongs.

Argument Type: String

`status`

Description: Deployment status of the inventory item.

Argument Type: String

Response

Returns an environment inventory item.

ec-perl

Syntax:

```
$<object>->createEnvironmentInventoryItem(<projectName>, <environmentName>, <app
licationName>, <componentName>,  <resourceName>, {<optionals>});
```

Example:

```
$ec->createEnvironmentInventoryItem("Default", "Web Server", "Deploy", "WAR fil
e", "local", {artifactName =>"com.ec:myApp", artifactVersion=>"3.1", description
=> "Required"});
```

ectool

Syntax:

```
ectool createEnvironmentInventoryItem <projectName> <environmentName> <applicati
onName> <componentName> <resourceName> [optionals]
```

Example:

```
ectool createEnvironmentInventoryItem "Default" "Web Server" "Deploy" "WAR file"
"local" -- artifactName "com.ec:myApp" --artifactVersion "3.1" --description "Re
quired"
```

Back to Top

createReservation

Creates a new reservation.

Required Arguments

`projectName`

Description: The name for the project that must be unique among all projects.

Argument Type: String

`reservationName`

Description: The name of the environment reservation.

Argument Type: String

Optional Arguments

`applicationName`

Description: The name of the application.

Argument Type: String

`beginDate`

Description: The date when the reservation begins.

Argument Type: String

`blackout`

Description: <*Boolean flag* - `0|1|true|false`>
When this argument is set to `true` or `1`, no deployments are allowed to run during the specified time period.

Argument Type: Boolean

`description`

Description: Comment text describing this object; not interpreted at all by ElectricFlow.

Argument Type: String

`endDate`

Description: The date when reservation ends.

Argument Type: String

`environmentName`

Description: The name of the environment.

Argument Type: String

`environmentProjectName`

Description: The name of the environment project.

Argument Type: String

`environmentTierName`

Description: The name of the environment tier.

Argument Type: String

`monthDays`

Description: A list of numbers from 1 to 31 separated by spaces, indicating zero or more days of the month.

Argument Type: String

`overlap`

Description: <*Boolean flag* - `0|1|true|false`>
When this argument is set to `true` or `1`, this reservation can overlap with another reservation. By default, reservations are not overlapped.

Argument Type: Boolean

`pipelineName`

Description: The name of the pipeline.

Argument Type: String

`recurrence`

Description: <*Boolean flag* - `0|1|true|false`> Recurrence reservation
When this argument is set to `true` or `1`, the reservation is a recurring reservation.

Argument Type: Boolean

`recurrenceEndDate`

Description: The date when the recurring reservation ends.

Argument Type: String

`releaseName`

Description: The name of the release.

Argument Type: String

`timeZone`

Description: The time zone to use when interpreting times.

Argument Type: String

`weekDays`

Description: Restricts the schedule to specified days of the week. Specify days of the week separated by spaces. Use English names "Monday", "Tuesday".

Argument Type: String

Response

Returns an environment reservation object.

ec-perl

Syntax:

```
$<object>->createReservation(<projectName>, <reservationName>, {<optionals>});
```

Example:

```
$ec->createReservation('Default', 'Main branch', {environmentName => "Web Serve
r", applicationName => "Heat Clinic", beginDate => '22:00', endDate => '23:30',
timeZone => 'local', overlap => true, weekDays => 'Monday' 'Wednesday' 'Frida
y'});
```

ectool

Syntax:

```
ectool createReservation <projectName> <reservationName> [optionals]
```

Example:

```
ectool createReservation 'Default' 'Main branch' --environmentName "Web Server"
--applicationName "Heat Clinic" --beginDate '22:00' --endDate '23:30' --timeZone
'local' --overlap true --weekDays 'Monday' 'Wednesday' 'Friday'
```

Back to Top

deleteEnvironment

Deletes an environment.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

Description: Name of the environment that 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", "PROD");
```

ectool

Syntax:

```
ectool deleteEnvironment <projectName> <environmentName>
```

Example:

```
ectool deleteEnvironment "Default" "PROD"
```

Back to Top

deleteEnvironmentInventoryItem

Deletes an inventory item.

Required Arguments

`projectName`

Description: Name for the project that 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

`environmentProjectName`

Description: Name of the project to which the environment or environment template belongs.

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->deleteEnvironmentInventoryItem(<projectName>, <environmentName>, <app
licationName>, <componentName>, <resourceName>, {optionals});
```

Example:

```
$cmdr->deleteEnvironmentInventoryItem("Utilities", "PROD", "Deploy", "WAR file",
"Server 1", {environmentProjectName => "Default");
```

ectool

Syntax:

```
ectool deleteEnvironmentInventoryItem <projectName> <environmentName> <applicati
onName> <componentName> <resourceName> [optionals]
```

Example:

```
ectool deleteEnvironmentInventoryItem "Utilities" "PROD" "Deploy" "WAR file" "Se
rver 1" --environmentProjectName "Default"
```

Back to Top

deleteReservation

Deletes the specified reservation.

Required Arguments

`projectName`

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

Argument Type: String

`reservationName`

Description: The name of the environment reservation.

Argument Type: String

Optional Arguments

None

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->deleteReservation(<projectName>, <reservationName>);
```

Example:

```
$cmdr->deleteReservation("Default", "Main branch");
```

ectool

Syntax:

```
ectool deleteReservation <projectName> <reservationName>
```

Example:

```
ectool deleteReservation "Default" "Main branch"
```

Back to Top

getEnvironment

Retrieves an environment by name.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

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

Argument Type: String

Optional Arguments

None

Response

Returns an environment element.

ec-perl

Syntax:

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

Example:

```
$ec->getEnvironment("Default", "Test Server");
```

ectool

Syntax:

```
ectool getEnvironment <projectName> <environmentName>
```

Example:

```
ectool getEnvironment "Default" "Test Server"
```

Back to Top

getEnvironmentApplications

Retrieves a list of applications installed on the given environment.

`projectName`

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

Argument Type: String

`environmentName`

Description: Name of the environment.

Argument Type: String

Optional Arguments

None

Response

Returns a list of applications for the specified environment.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getEnvironmentApplications "Default" "Test Server"
```

Back to Top

getEnvironmentInventory

Retrieves a per-component grouped list of inventory items.

Required Arguments

`projectName`

Description: Name for the project that 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

`applicationEntityRevisionId`

Description: Revision ID of the versioned object.

Argument Type: UUID

`environmentProjectName`

Description: Name for the project to which the environment or environment template belongs.

Argument Type: String

Response

Returns a per-component grouped list of inventory items.

ec-perl

Syntax:

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

Example:

```
$ec->getEnvironmentInventory("Default", "Test Server", "Save snapshot");
```

ectool

Syntax:

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

Example:

```
ectool getEnvironmentInventory "Default" "Test Server" "Save snapshot"
```

Back to Top

getEnvironmentInventoryItem

Retrieves an inventory item.

Required Arguments

`projectName`

Description: Name for the project that 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

`environmentProjectName`

Description: Name for the project to which the environment or environment template belongs.

Argument Type: String

`includeResourceDetails`

Description: <*Boolean flag* - `0|1|true|false`>
When this argument is set to `true` or `1`, the response includes the resource details. The default is `false` or `0`.

Argument Type: Boolean

Response

Retrieves an inventory item.

ec-perl

Syntax:

```
$<object>->getEnvironmentInventoryItem(<projectName>, <environmentName>, <applic
ationName>, <componentName>, <resourceName>, {<optionals>});
```

Example:

```
$ec->getEnvironmentInventoryItem("Default", "Test Lab", "Save snapshot", "WAR fi
le", "QA server", {environmentProjectName => "QA only"});
```

ectool

Syntax:

```
ectool getEnvironmentInventoryItem <projectName> <environmentName> <applicationN
ame> <componentName> <resourceName> [optionals]
```

Example:

```
ectool getEnvironmentInventoryItem "Default" "Test Lab" "Save snapshot" "WAR fil
e" "QA server" --environmentProjectName "QA only"
```

Back to Top

getEnvironmentInventoryItems

Retrieves all the inventory items for a given environment.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

Description: Name of the environment.

Argument Type: String

Optional Arguments

`includeResourceDetails`

Description: <*Boolean flag* - `0|1|true|false`>
When this argument is set to `true` or `1`, the response includes the resource details. The default is `false` or `0`.

Argument Type: Boolean

Response

Returns all inventory items for the specified environment.

ec-perl

Syntax:

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

Example:

```
$ec->getEnvironmentInventoryItems("Default", "Test Lab");
```

ectool

Syntax:

```
ectool getEnvironmentInventoryItems <projectName> <environmentName>
```

Example:

```
ectool getEnvironmentInventoryItems "Default" "Test Lab"
```

Back to Top

getEnvironments

Retrieves all environments in a project.

Required Arguments

`projectName`

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

Argument Type: String

Optional Arguments

None

Response

Returns zero or more environment elements.

ec-perl

Syntax:

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

Example:

```
$ec->getEnvironments("Default");
```

ectool

Syntax:

```
ectool getEnvironments <projectName>
```

Example:

```
ectool getEnvironments "Default"
```

Back to Top

getReservation

Retrieves an environment reservation by its name.

Required Arguments

`projectName`

Description: The name of the project that must be unique among all projects.

Argument Type: String

`reservationName`

Description: The name of the environment reservation.

Argument Type: String

Optional Arguments

`beginDate`

Description: The date when the reservation begins.

Argument Type: String

`endDate`

Description: The date when the reservation ends.

Argument Type: String

`includeTimeslots`

Description: <*Boolean flag* - `0|1|true|false`>
When this argument is set to `true` or `1`, time slots are included in the reservation response.

Argument Type: Boolean

`timeZone`

Description: The time zone to use when interpreting times.

Argument Type: String

Response

Returns zero or more reservation objects.

ec-perl

Syntax:

```
$<object>->getReservation(<projectName>, <reservationName>, {<optionals>});
```

Example:

```
$ec->getReservation("Default", "Main branch", {includeTimeslots => true, timeZon
e => "local"} );
```

ectool

Syntax:

```
ectool getReservation <projectName> <reservationName> [optionals]
```

Example:

```
ectool getReservation "Default" "Main branch" --includeTimeslots true --timeZone
"local"
```

Back to Top

getReservations

Retrieves all the environment reservations.

Required Arguments

None

Optional Arguments

`applicationName`

Description: The name of the application.

Argument Type: String

`beginDate`

Description: The date when the reservation begins.

Argument Type: String

`blackout`

Description: <*Boolean flag* - `0|1|true|false`>
When this argument is set to `true` or `1`, no deployments are allowed to run during the specified time period

Argument Type: Boolean

`endDate`

Description: The date when the reservation ends.

Argument Type: String

`environmentName`

Description: The name of the environment.

Argument Type: String

`environmentProjectName`

Description: The name of the environment project.

Argument Type: String

`environmentTierName`

Description: The name of the environment tier.

Argument Type: String

`includeTimeslots`

Description: <*Boolean flag* - `0|1|true|false`>
When this argument is set to `true` or `1`, time slots are included in the response.

Argument Type: Boolean

`pipelineName`

Description: The name of the pipeline.

Argument Type: String

`projectName`

Description: The name for the project that must be unique among all projects.

Argument Type: String

`releaseName`

Description: The name of the release.

Argument Type: String

`reserverTypes`

Description: The types of the objects reserving the environments (the Reserver).

Argument Type: Collection

`timeZone`

Description: The time zone to use when interpreting times.

Argument Type: String

Response

Returns zero or more reservation objects.

ec-perl

Syntax:

```
$<object>->getReservations({<optionals>});
```

Example:

```
$ec->getReservations({environmentName => "PROD", pipelineName => "RC build", inc
ludeTimeslots => true, timeZone => "local"});
```

ectool

Syntax:

```
ectool getReservations [optionals]
```

Example:

```
ectool getReservations --environmentName "PROD" --pipelineName => "RC build" --i
ncludeTimeslots true --timeZone "local"
```

Back to Top

getRunSchedules

Retrieves the run schedules with environment reservations.

Required Arguments

None

Optional Arguments

`beginDate`

Description: The date when the reservation begins. The format is <*yyyy-mm-dd*>.

Argument Type: String

`endDate`

Description: The date when the reservation ends. The format is <*yyyy-mm-dd*>.

Argument Type: String

`filters`

Description: Filters to use during the search operation.

Argument Type: Collection

`includeTimeslots`

Description: <*Boolean flag* - `0|1|true|false`>
If this is set to `true` or `1`, the response includes the resource count.

Argument Type: Boolean

`timeZone`

Description: The time zone to use when interpreting times.

Argument Type: String

Response

Returns zero or more schedules with environment reservations.

ec-perl

Syntax:

```
$<object>->getRunSchedules({<optionals>});
```

Example:

```
$ec->getRunSchedules({beginDate => "2016-07-01", endDate => "2016-09-01", includ
eTimeslots => true, timeZone => "local"});
```

ectool

Syntax:

```
ectool getRunSchedules [optionals]
```

Example:

```
ectool getRunSchedules --beginDate "2016-07-01" --endDate => "2016-09-01" --incl
udeTimeslots true --timeZone "local"
```

Back to Top

modifyEnvironment

Modifies an environment.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

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

Argument Type: String

Optional Arguments

`description`

Description: Comment text describing this object; not interpreted at all by ElectricFlow.

Argument Type: String

`environmentEnabled`

Description: True to enable the environment.

Argument Type: Boolean

`newName`

Description: New name for the environment.

Argument Type: String

`reservationRequired`

Description: <*Boolean flag* -`0|1|true|false`>
Set this flag to "true" to allow deployments to run using only using environment reservations.

Argument Type: Boolean

`rollingDeployEnabled`

Description: <*Boolean flag* -`0|1|true|false`>
Set this flag to "true" or "1" to enable the environment.

Argument Type: Boolean

`rollingDeployType`

Description: The type of rolling deploy supported by the environment.

Argument Type: RollingDeployType

Response

Returns an updated environment element.

ec-perl

Syntax:

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

Example:

```
$ec->modifyEnvironment("Default", "Pet Store", {newName => "New Pet Store", desc
ription => "Pet store website", rollingDeployEnabled => true});
```

ectool

Syntax:

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

Example:

```
ectool modifyEnvironment "Default" "Pet Store" --newName "New Pet Store" --descr
iption "Pet store website" --rollingDeployEnabled true
```

Back to Top

modifyEnvironmentInventoryItem

Modifies an existing environment inventory item.

Required Arguments

`projectName`

Description: Name for the project that 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

`applicationTierName`

Description: Name of the application tier.

Argument Type: String

`artifactName`

Description: Name of the artifact for the inventory item.

Argument Type: String

`artifactSource`

Description: Source of the artifact.

Argument Type: String

`artifactUrl`

Description: URL of the artifact.

Argument Type: String

`artifactVersion`

Description: Version of the artifact for the inventory item.

Argument Type: String

`description`

Description: Comment text describing this object; not interpreted by ElectricFlow.

Argument Type: String

`environmentProjectName`

Description: Name for the project to which the environment or environment template belongs.

Argument Type: String

`status`

Description: Inventory deployment status.

Argument Type: JobOutcome

Response

Returns an updated environment inventory item.

ec-perl

Syntax:

```
$<object>->modifyEnvironmentInventoryItem(<projectName>, <environmentName>, <app
licationName>, <componentName>, <resourceName>, {<optionals>});
```

Example:

```
$ec->modifyEnvironmentInventoryItem("Default", "PROD", "Deploy", "WAR file", "Se
rver 1", {artifactVersion => "V3"});
```

ectool

Syntax:

```
ectool modifyEnvironmentInventoryItem <projectName> <environmentName> <applicati
onName> <componentName> <resourceName> [optionals]
```

Example:

```
ectool modifyEnvironmentInventoryItem "Default" "PROD" "Deploy" "WAR file" "Serv
er 1" --artifactVersion "V3"
```

Back to Top

modifyReservation

Modifies an environment reservation.

Required Arguments

`projectName`

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

Argument Type: String

`reservationName`

Description: The name of the environment reservation.

Argument Type: String

Optional Arguments

`applicationName`

Description: The name of the application

Argument Type: String

`beginDate`

Description: The date when the reservation begins.

Argument Type: String

`blackout`

Description: <*Boolean flag* - `0|1|true|false`>
When this argument is set to `true` or `1`, no deployments are allowed to run during the specified time period.

Argument Type: Boolean

`description`

Description: Comment text describing this object; not interpreted at all by ElectricFlow.

Argument Type: String

`endDate`

Description: The date when the reservation ends.

Argument Type: String

`environmentName`

Description: The name of the environment.

Argument Type: String

`environmentProjectName`

Description: The name of the environment project.

Argument Type: String

`environmentTierName`

Description: The name of the environment tier.

Argument Type: String

`monthDays`

Description: A list of numbers from 1 to 31 separated by spaces, indicating zero or more days of the month.

Argument Type: String

`newName`

Description: New name for an existing object.

Argument Type: String

`overlap`

Description: <*Boolean flag* - `0|1|true|false`>
When this argument is set to `true` or `1`, this reservation can overlap with another reservation. By default, reservations are not overlapped.

Argument Type: Boolean

`pipelineName`

Description: The name of the pipeline.

Argument Type: String

`recurrence`

Description: <*Boolean flag* - `0|1|true|false`> Recurrence reservation
When this argument is set to `true` or `1`, the reservation is a recurring reservation.

Argument Type: Boolean

`recurrenceEndDate`

Description: The date when the recurring reservation ends.

Argument Type: String

`releaseName`

Description: The name of the release.

Argument Type: String

`timeZone`

Description: The time zone to use when interpreting times.

Argument Type: String

`weekDays`

Description: Restricts the schedule to specified days of the week. Specify days of the week separated by spaces. Use English names "Monday", "Tuesday".

Argument Type: String

Response

Returns a modified environment reservation object.

ec-perl

Syntax:

```
$<object>->modifyReservation(<projectName>, <reservationName>, {<optionals>});
```

Example:

```
$ec->modifyReservation('Default', 'Main branch', {environmentName => "Web Serve
r", applicationName => "Heat Clinic", beginDate => '22:00', endDate => '23:30',
timeZone => 'local', overlap => false, weekDays => 'Monday' 'Wednesday' 'Friday'
'Sunday'});
```

ectool

Syntax:

```
ectool modifyReservation <projectName> <reservationName> [optionals]
```

Example:

```
ectool modifyReservation 'Default' 'Main branch' --environmentName "Web Server"
--applicationName "Heat Clinic" --beginDate '22:00' --endDate '23:30' --timeZone
'local' --overlap true --weekDays 'Monday' 'Wednesday' 'Friday' 'Sunday'
```

Back to Top

seedEnvironmentInventory

Creates a new environment.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

Description: Name of the environment.

Argument Type: String

`applicationName`

Description: Application that owns the inventory item.

Argument Type: String

`applicationProcessName`

Description: Application process that will be deployed to the environment.

Argument Type: String

`status`

Description: Deployment status of the environment inventory.

Argument Type: JobOutcome

Optional Arguments

`artifactName`

Description: Artifact name for the inventory item.

Argument Type: String

`artifactSource`

Description: Source of the artifact.

Argument Type: String

`artifactUrl`

Description: URL of the artifact.

Argument Type: String

`artifactVersion`

Description: Artifact version for the inventory item.

Argument Type: String

`componentName`

Description: Component that owns the inventory item

Argument Type: String

`environmentProjectName`

Description: Name for the project to which the environment or environment template belongs.

Argument Type: String

`resourceNames`

Description: Resources assigned to the environment.

Argument Type: Collection

`snapshotName`

Description: Name of the snapshot.

Argument Type: String

Response

Returns a new environment inventory item.

ec-perl

Syntax:

```
$<object>->seedEnvironmentInventory(<projectName>, <environmentName>, <applicati
onName>, <applicationProcessName>, <status>, {<optionals>});
```

Example:

```
$ec->seedEnvironmentInventory("Default", "PROD", "Deploy", "Clean up DB", "succe
ss", {artifactVersion => "3.0"});
```

ectool

Syntax:

```
ectool seedEnvironmentInventory <projectName> <environmentName> <applicationNam
e> <applicationProcessName> <status> [optionals]
```

Example:

```
ectool seedEnvironmentInventory "Default" "PROD" "Deploy" "Clean up DB" "succes
s" --artifactVersion "3.0"
```

Back to Top

API Commands - Environment Tier

addResourcesToEnvironmentTier

Adds resources to the specified environment tier.

Required Arguments

`projectName`

Description: Name for the project that 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.

Argument Type: String

`environmentTierName`

Description: Name for the environment tier that must be unique among all tiers in the environment.

Argument Type: String

Optional Arguments

`resourceNames`

Description: List of resources to add to the environment tier.

Argument Type: Collection

`resourcePhaseMappings`

Description: A map of resources and rolling deploy phases.

Argument Type: Map

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->addResourcesToEnvironmentTier(<projectName>, <environmentName>, <envi
ronmentTierName>, {<optionals>});
```

Example:

```
$ec->addResourcesToEnvironmentTier("Default", "QA", "Tomcat", {resourceNames =>
"local"});
```

ectool

Syntax:

```
ectool addResourcesToEnvironmentTier <projectName> <environmentName> <environmen
tTierName> [optionals]
```

Example:

```
ectool addResourcesToEnvironmentTier "Default" "QA" "Tomcat" --resourceNames "lo
cal"
```

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

`batchSize`

Description: The size of the batch supported by the environment tier.

Argument Type: String

`batchSizeType`

Description: The type of batch size supported by the environment tier.

Argument Type: BatchSizeType

`description`

Description: Comment text describing this object; not interpreted at all by ElectricFlow.

Argument Type: String

`resourceNames`

Description: List of resources to add to the environment tier.

Argument Type: Collection

`resourcePhaseMappings`

Description: A map of the resources to the rolling deploy phases to which they are assigned.

Argument Type: Map

`resourcePoolNames`

Description: A list of resource pools to add to the environment tier.

Argument Type: Collection

`resourcePoolPhaseMappings`

Description: A map of the resource pools to the rolling deploy phases to which they are assigned.

Argument Type: Map

Response

Returns an environment tier element.

ec-perl

Syntax:

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

Example:

```
$ec->createEnvironmentTier("Default", "PROD", "Web Services", {description => "R
equired for AWS"});
```

ectool

Syntax:

```
ectool createEnvironmentTier <projectName> <environmentName> <environmentTierNam
e> [optionals]
```

Example:

deleteEnvironmentTier

Deletes an environment tier.

Required Arguments

`projectName`

Description: Name of the project that 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. The name must be unique among all projects.

Argument Type: String

`environmentTierName`

Description: Name of the environment tier that 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>, <environmentT
ierName>);
```

Example:

```
$ec->deleteEnvironmentTier("Default", "PROD", "App Server");
```

ectool

Syntax:

```
ectool deleteEnvironmentTier <projectName> <environmentName> <environmentTierNam
e>
```

Example:

```
ectool deleteEnvironmentTier "Default" "PROD" "App Server"
```

getEnvironmentTier

Retrieves an environment tier by name.

Required Arguments

`projectName`

Description: Name of the project that 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. It must be unique among all projects.

Argument Type: String

`environmentTierName`

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

Argument Type: String

Optional Arguments

None

Response

Returns an environment tier element.

ec-perl

Syntax:

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

Example:

```
$ec->getEnvironmentTier("Default", "QA Lab", "Test Machine 1");
```

ectool

Syntax:

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

Example:

```
ectool getEnvironmentTier "Default" "QA Lab" "Test Machine 1"
```

getEnvironmentTiers

Retrieves all environment tiers in an environment.

Required Arguments

`projectName`

Description: Name of the project that 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. It must be unique among all projects.

Argument Type: String

Optional Arguments

None

Response

Returns zero or more environment tier elements.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getEnvironmentTiers "Default" "QA Lab"
```

modifyEnvironmentTier

Modifies an environment tier.

Required Arguments

`projectName`

Description: Name of the project that 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.

Argument Type: String

`environmentTierName`

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

Argument Type: String

Optional Arguments

`batchSize`

Description: The size of the batch supported by the environment tier.

Argument Type: String

`batchSizeType`

Description: The type of batch size supported by the environment tier.

Argument Type: BatchSizeType

`description`

Description: Comment text describing this object; not interpreted at all by ElectricFlow.

Argument Type: String

`newName`

Description: New name for an existing environment tier.

Argument Type: String

`resourceNames`

Description: List of resources to add to the environment tier.

Argument Type: Collection

`resourcePhaseMappings`

Description: A map of the resources to the rolling deploy phases to which they are assigned.

Argument Type: Map

`resourcePoolNames`

Description: A list of resource pools to add to the environment tier.

Argument Type: Collection

`resourcePoolPhaseMappings`

Description: A map of the resource pools to the rolling deploy phases to which they are assigned.

Argument Type: Map

Response

Retrieves an updated environment tier element.

ec-perl

Syntax:

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

Example:

```
$ec->modifyEnvironmentTier("Default", "QA lab", "Tomcat 1", {newName => "Tomcat
2", description => "New server});
```

ectool

Syntax:

```
ectool modifyEnvironmentTier <projectName> <environmentName> <environmentTierNam
e> [optionals]
```

Example:

```
ectool modifyEnvironmentTier "Default" "QA lab" "Tomcat 1" --newName "Tomcat 2"
--description "New server"
```

removeResourcesFromEnvironmentTier

Removes the given resources from the given environment tier.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

Description: The name of the environment.

Argument Type: String

`environmentTierName`

Description: Name for the environment tier that must be unique among all tiers for the environment.

Argument Type: String

Optional Arguments

`resourceNames`

Description: List of resources to remove from the environment tier.

Argument Type: Collection

Response

Removes zero or more environment tier elements.

ec-perl

Syntax:

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

Example:

```
$ec->removeResourcesFromEnvironmentTier("Default", "Production", "Apache");
```

ectool

Syntax:

```
ectool removeResourcesFromEnvironmentTier <projectName> <environmentName> <envir
onmentTierName> [optionals]
```

Example:

```
ectool removeResourcesFromEnvironmentTier "Default" "Production" "Apache"
```

API Commands - Gateway and Zone 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
`gatewayName`	The name of the gateway that must be unique among all gateway names. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. Argument type: String

Arguments	Descriptions
`gatewayDisabled`	(Optional) <*Boolean flag* - `0|1|true|false` > If set to **1** or "true", the gateway is disabled. Argument type: Boolean
`hostName1`	(Optional) The domain name or IP of the agent 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 the *Resource1* definition. Argument type: String
`hostName2`	(Optional) The domain name or IP of the agent 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 the *Resource2* definition. Argument type: String
`port1`	(Optional) The port number used by *Resource1*. The default is the port number used by the resource. Argument type: Integer
`port2`	(Optional) The port number used by *Resource2*. The default is the port number used by the resource. Argument type: Integer
`resourceName1`	(Optional) Name of first resource in the gateway specification. Do not include spaces in the resource name. Other resources in this resource's zone forward messages through this resource to agents in the `resourceName2` zone. Argument type: String
`resourceName2`	(Optional) Name of second resource in the gateway specification. Do not include spaces in the resource name. Other resources in this resource's zone forward messages through this resource to agents in the `resourceName1` zone. Argument type: String

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 Zone
B", resourceName1 => "ResourceA", resourceName2 => "ResourceB"});
```

ectool

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

Example

```
ectool createGateway AB_Gateway --description "Gateway linking ZoneA and ZoneB" --r
esourceName1 "ResourceA" --resourceName2 "ResourceB"
```

Back to Top

deleteGateway

Deletes a gateway.

You must enter a `gatewayName`.

Arguments	Descriptions
`gatewayName`	The name of the gateway to delete. Argument type: String

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. Argument type: String

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
`gatewayName`	The name of the gateway. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. Argument type: String
`gatewayDisabled`	(Optional) <*Boolean flag* - `0|1|true|false` > If this is set to `1` or `true`, the gateway is disabled. Argument type: Boolean
`hostName1`	(Optional) The domain name or IP address of the agent 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 the *Resource1* definition. Argument type: String
`hostName2`	(Optional) The domain name or IP address of the agent 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 the *Resource2* definition. Argument type: String

Arguments	Descriptions
`newName`	(Optional) New name of the gateway. Argument type: String
`port1`	(Optional) The port number used by *Resource1*. The default is the port number used by the resource. Argument type: Integer
`port2`	(Optional) The port number used by *Resource2*. The default is the port number used by the resource. Argument type: Integer
`resourceName1`	(Optional) Name of first resource in the gateway specification. Do not include spaces in the resource name. Other resources in this resource's zone forward messages through this resource to agents in the `resourceName2` zone. Argument type: String
`resourceName2`	(Optional) Name of second resource in the gateway specification. Do not include spaces in the resource name. Other resources in this resource's zone forward messages through this resource to agents in the `resourceName1` zone. Argument type: String

Positional arguments

`gatewayName`

Response

An updated `gateway` object.

ec-perl

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

Example

```
$cmdr->modifyGateway ("AB_Gateway", {description=> "Gateway linking zoneA and zone
B", 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
`zoneName`	The unique name of the zone. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. Argument type: String

Positional arguments

`zoneName`

Response

Returns a `zone` object.

ec-perl

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

Example

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

ectool

syntax: `ectool createZone <zoneName> [optionals]`

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. Argument type: String

Positional arguments

`zoneName`

Response

None

ec-perl

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

Example

```
$cmdr->deleteZone ("Dev Tests");
```

ectool

syntax: `ectool deleteZone <zoneName>`

Example

```
ectool deleteZone "Dev Tests"
```

Back to Top

getZone

Finds a zone by name.

You must specify a `zoneName`.

Arguments	Descriptions
`zoneName`	The name of the zone. Argument type: String

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 ("Dev Zone");
```

ectool

syntax: `ectool getZone <zoneName>`

Example

```
ectool getZone "Dev Zone"
```

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
`zoneName`	The name of this zone. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. Argument type: String
`newName`	(Optional) New name of the zone. Argument type: String

Positional arguments

`zoneName`

Response

Returns an updated zone element.

ec-perl

syntax: `$cmdr->modifyZone (<zoneName>, {<optionals>});`

Example

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

ectool

syntax: `ectool modifyZone <zoneName> [optionals]`

Example

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

Back to Top

API Commands - Installing or Upgrading Remote Agents

For information about the API commands that are used to install or upgrade remote agents, see the "Installing or Upgrading Remote Agents" section in the "Automation Platform" chapter of the *ElectricFlow User Guide*.

Back to Top

API Commands - Job Management

abortAllJobs

Aborts all running jobs.

Arguments	Descriptions
`force`	(Optional) <*Boolean flag* - `0|1|true|false` > If this is set to `1` or `true`, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, executing steps marked “always run". Argument type: Boolean
`reason`	(Optional) 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, which is similar to a text description. Argument type: String

Positional arguments

None

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool abortAllJobs [optionals]`

Example

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

Back to Top

abortJob

Aborts a running job.

You must enter a `jobId`.

Arguments	Descriptions
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID

Arguments	Descriptions
`force`	(Optional) <*Boolean flag -* `0|1|true|false`> If this is set to `1` or *true*, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, executing steps marked "always run". Argument type: Boolean
`reason`	(Optional) 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, which is similar to a text description. Argument type: String

Positional arguments

`jobId`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool abortJob <jobId> [optionals]`

Example

```
ectool abortJob 4fa765dd-73f1-11e3-b67e-b0a420524153 --force 1
```

Back to Top

abortJobStep

Aborts any type of running step, including a command step or subprocedure step.

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

You must enter a `jobStepId`.

Arguments	Descriptions
`jobStepId`	The unique ElectricFlow-generated identifier (a UUID) for a job step that is assigned automatically when the job step is created. The system also accepts a job step name assigned to the job step by its name template. Argument type: UUID
`force`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to *1* or *true*, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, for example, executing steps marked "always run". Argument type: Boolean
`reason`	(Optional) 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, which is similar to a text description. Argument type: String

Positional arguments

`jobStepId`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool abortJobStep <jobStepId> [optionals]`

Example

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

Back to Top

deleteJob

Deletes a job from the ElectricFlow database.

You must specify a `jobId`.

Arguments	Descriptions
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`waitForDeleteToComplete`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to *1* or *true*, ElectricFlow deletes the jobs after completing it. If the value is `0` or `false`, the job is deleted immediately. Argument type: Boolean

Positional arguments

`jobId`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteJob(4fa765dd-73f1-11e3-b67e-b0a420524153, {waitForDeleteToComplete =>
0});
```

ectool

syntax: `ectool deleteJob <jobId> [<optionals>]`

Example

```
ectool deleteJob 4fa765dd-73f1-11e3-b67e-b0a420524153 --waitForDeleteToComplete 0
```

Back to Top

findJobSteps

Returns a list of job steps from a single job or from a single subprocedure job step. This command is used by the
Job Details web page in the ElectricFlow 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 ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	The unique ElectricFlow-generated identifier (a UUID) for a job step that is assigned automatically when the job step is created. The system also accepts a job step name assigned to the job step by its name template. Argument type: UUID
`filters`	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. Argument type: Collection
`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. Argument type: Integer
`selects`	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 ElectricFlow Perl API. Argument type: Collection

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 ElectricFlow:\n" .
               $xPath-> findnodes_as_string("/"). "\n";
```

Example 2

```
my $xPath = $cmdr->findJobSteps({jobStepId  => "5da765dd-73f1-11e3-b67e-b0a42052415
3"});
   print "Return data from ElectricFlow:\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 that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`structureOnly`	(Optional) <*Boolean flag -* `0|1|true|false`> If this is set to `1` or `true`, the amount of information returned is reduced to minimal structural information. Argument type: Boolean

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> [optionals]`

Example

```
ectool getJobDetails 4fa765dd-73f1-11e3-b67e-b0a420524153 --structureOnly 1
```

Back to Top

getJobInfo

Retrieves all information about a job, *except* for job step information.

You must specify a `jobId`.

Arguments	Descriptions
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID

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 that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID

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:
If you use `sortKey` or `sortOrder`, you must use both arguments together.
You can use `firstResult` and `maxResults` together or separately to select a limited sublist of jobs for the result set.

Arguments	Descriptions
`firstResult`	The first row to return in the results. This is the <*index number*> 0-based index that identifies the first element returned from filtered, sorted result set. Argument type: Integer

Arguments	Descriptions
`maxResults`	This number, *<result count>*, sets the maximum number of returned jobs. Argument type: Integer
`sortKey`	How to sort the results: `<jobId|jobName|start|finish|procedureName>`. Argument type: String
`sortOrder`	The order in which to sort the results: `<ascending|descending>`. Argument type: SortOrder
`status`	Filter jobs by this status: `<running|completed|runnable>`. Argument type: String

Positional arguments

None

Response

One or more `job` elements. A `job` element contains summary information only.

ec-perl

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

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 [optionals]`

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 for the project that must be unique among all projects. Argument type: String
`scheduleName`	The name for the schedule that must be unique among all schedules for the project. Argument type: String

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('QA Tests', 'Required');
```

ectool

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

Example

```
ectool getJobsForSchedule 'QA Tests' 'Required'
```

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 that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID

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 ElectricFlow-generated identifier (a UUID) for a job step that is assigned automatically when the job step is created. The system also accepts a job step name assigned to the job step by its name template. Argument type: UUID
`structureOnly`	(Optional) <*Boolean flag* - `<0|1|true|false>` c`true`, the amount of information returned is reduced to minimal structural information. Argument type: Boolean

Positional arguments

`jobStepId`

Response

A `jobStep` element.

ec-perl

syntax: `$cmdr->getJobStepDetails(<jobStepId>, {<optionals>});`

Example

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

ectool

syntax: `ectool getJobStepDetails <jobStepId> [optionals]`

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 ElectricFlow-generated identifier (a UUID) for a job step that is assigned automatically when the job step is created. The system also accepts a job step name assigned to the job step by its name template. Argument type: UUID

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

getJobSummaries

Retrieves summary information about jobs.

Arguments	Descriptions

`filters`	(Optional) 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 ElectricFlow 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 ElectricFlow 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)`
	
	Argument type: Collection

Arguments	Descriptions
`includeLastSuccess`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, the step will include the last successful job if it was not already included. Argument type: Boolean
`maxFailedSteps`	(Optional) Maximum number of failed steps to return. Argument type: Integer
`maxJobs`	(Optional) Maximum number of jobs to return. Argument type: Integer
`maxRunningSteps`	(Optional) Maximum number of running steps to return. Argument type: Integer

Positional arguments

None

Response

A summary of the job steps with the specified properties.

ec-perl

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

Example

```
$cmdr->getJobSummaries ({maxFailedSteps => 6});
```

ectool

syntax:`ectool getJobSummaries [optionals]`

Example

```
ectool getJobSummaries --maxFailedSteps 6
```

Back to Top

getJobSummary

Retrieves a job and its job steps with only the specified job and job step properties.

Arguments	Descriptions
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobProperties`	(Optional) A comma-separated list of intrinsic job properties to include in the output. Argument type: String
`jobStepProperties`	(Optional) A comma-separated list of intrinsic job step properties to include in the output. Argument type: String

Positional arguments

None.

Response

A summary of the job step with the specified properties.

ec-perl

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

Example

```
$cmdr->getJobSummary ({jobId => 5da765dd-73f1-11e3-b67e-b0a420524153});
```

ectool

syntax:`ectool getJobSummary [optionals]`

Example

```
ectool getJobSummary --jobId 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`	Name of the project that contains the jobs you want to move. Argument type: String
`destinationProject`	New project that will contain the jobs. Argument type: String

Positional arguments

`sourceProject, destinationProject`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->moveJobs(<sourceProject>, <destinationProject>);`

Example

```
$cmdr->moveJobs("Default", "Utilities");
```

ectool

syntax: `ectool moveJobs <sourceProject> <destinationProject>`

Example

```
ectool moveJobs "Default" "Utilities"
```

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. Wait until the job completes. 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`.

Note: The `pollInterval` and *timeout* 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 time out in 60 seconds.

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`actualParameter`	(Optional) 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`. Argument type: Map
`credentialName`	(Optional) `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
`credentials`	(Optional) Use the following syntax to specify a credential: `<credName>=<userName> [<credName>=<userName> ...]].` Argument type: Collection
`destinationProject`	(Optional) This argument is used to specify the name of the destination project. Argument type: String
`password`	(Optional) The password for the user running the procedure. Argument type: String
`pollInterval`	(Optional) This argument requires setting a value in *seconds* to determine how often ectool queries the ElectricFlow 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. If this option is not specified, `runProcedure` returns immediately. If it is specified, `runProcedure` waits until the job completes. Argument type: Integer
`priority`	(Optional) 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`. Argument type: JobPriority
`procedureName`	(Optional) The name for the procedure; must be unique within the project. Argument type: String
`scheduleName`	(Optional) The name for the schedule; must be unique among all schedules for the project. Use this option if you want to use the parameters from an existing specific schedule. Argument type: String

Arguments	Descriptions
`timeout`	(Optional) This argument requires a value set in *seconds*. If `pollInterva` 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. Argument type: Integer
`userName`	(Optional) The name of the user who is running the procedure. Argument type: String

Positional arguments

`projectName`

Response

The new job ID.

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> [optionals]`

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 ElectricFlow step.

A job cannot be renamed after it has completed.

Arguments	Descriptions
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`newName`	New name of the job. Argument type: String

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

ElectricFlow includes a powerful built-in scheduler for both managing execution and real-time reporting for a "running" process. Most ElectricFlow 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 ElectricFlow excels—especially as it applies to monitoring the status of complex processes and workflows as they progress in real-time through the ElectricFlow system. The ElectricFlow GUI also provides powerful auditing capabilities for reviewing results of complex process runs.

External Job APIs are designed to leverage the ElectricFlow 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 ElectricFlow Jobs page. ElectricFlow users and the external scheduler can then monitor the complete integrated system through a single interface—the ElectricFlow 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 ElectricFlow as a job.

Using the API does NOT consume agent resources. The API simply allows for graphical representation of external jobs within ElectricFlow.

completeJob

Completes an externally managed job. Marks the job root step so the job is marked "completed" when all child steps are completed, and updates the runtime for the root step.

You must specify a `jobId`.

Arguments	Descriptions
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`force`	(Optional) <*Boolean flag* - `0|1|true|false`> If true, all external steps belonging to the job will be marked "complete". This arguments 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. Argument type: Boolean

Arguments	Descriptions
`outcome`	(Optional) Overall outcome for a job or step: `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. Argument type: JobOutcome

Positional arguments

`jobId`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool completeJob <jobId> [optionals]`

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 runtime.

You must specify a `jobStepId`.

Arguments	Descriptions
`jobStepId`	The unique ElectricFlow-generated identifier (a UUID) for a job step that is assigned automatically when the job step is created. The system also accepts a job step name assigned to the job step by its name template. Argument type: UUID
`exitCode`	(Optional) The exit code of a job step. Argument type: Integer
`force`	(Optional) <*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. Argument type: Boolean
`outcome`	(Optional) Overall outcome for a job or step: `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. Argument type: JobOutcome

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
`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. Argument type: String
`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. Argument type: String
`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. Argument type: String
`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. Argument type: String

Arguments	Descriptions
`status`	The starting status for the job:`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. Argument type: JobStatus

Positional arguments

`projectName` or `destinationProject`

Response

The new job ID.

ec-perl

syntax: `$cmdr->createJob(<projectName or destinationProject>, {<optionals>});`

Example

```
$cmdr->createJob(projectName => "Default", {status => "Scheduled"});
```

ectool

syntax: `ectool createJob <projectName or destinationProject> [optionals]`

Example

```
ectool createJob "Default" --status "Scheduled"
```

Back to Top

createJobStep

Use this API to add ElectricFlow managed job steps to a running job or job step and 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`.

Note: External job steps do not participate in error handling processing. When you execute a job that includes a step with the `--errorhandling abortProcedureNow` argument, this argument is ignored if the step fails. The job then continues running the rest of the steps.

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. Argument type: Map
`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". Argument type: Boolean
`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". Argument type: Boolean
`command`	The command to run. This argument is applicable to command job steps only. Argument type: String
`comment`	The script to execute the functions in this step, which are passed to the step's shell for execution. Argument type: String
`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. Argument type: String

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. Argument type: String
`credentials`	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"topic in the ElectricFlow User Guide. Argument type: Collection
`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. Argument type: ErrorHandling
`exclusive`	<*Boolean flag* -`0|1|true|false`> If set to `1` or `true`, this job step should acquire and retain this resource exclusively. The default is `0` or `false`. **Note:** Setting `exclusive` sets `exclusiveMode` to `"job"`. Argument type: Boolean

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. Argument type: ExclusiveMode
`external`	If set, indicates this job step is an external step. ElectricFlow will not schedule or run agent commands for external steps, but instead, represents a step managed outside of ElectricFlow. 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". Argument type: Boolean
`jobStepId`	The unique ElectricFlow-generated identifier (UUID) for the parent job step that is assigned automatically when the job step is created. The system also accepts a job step name assigned to the job step by its name template. You can specify `parentPath` or `jobStepId` or both. If you specify both `parentPath` and `jobStepId`, `parentPath` is preferred. Argument type: UUID
`jobStepName`	The name of the new dynamic job step to create. Argument type: String
`logFileName`	A custom log file name produced by running the job step. By default, ElectricFlow assigns a unique name for this file. Argument type: String
`parallel`	<*Boolean flag* -`0|1|true|false`> If set to `1` or `true`, this job step should run at the same time as adjacent job steps marked to run as parallel. The default is `0` or `false`. Argument type: Boolean

Arguments	Descriptions
`parentPath`	The path of the parent job step. This argument determines the parent job for the new `jobStep`. The new `jobStep` is executed in context of the parent job step. You can specify `parentPath` or `jobStepId` or both. If you specify both `parentPath` and `jobStepId`, `parentPath` is preferred. Argument type: String
`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. ElectricFlow 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). Argument type: String

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 fixed text or text embedding a 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. Argument type: String 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 for an existing step definition that will be copied to the new job step. Argument type: String

Arguments	Descriptions
`projectName`	The name of the project for an existing step definition that will be copied to the new job step. Argument type: String
`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`. Argument type: Boolean
`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. • `releaseToJob` - allows a job step to promote a "step exclusive" resource to a Job exclusive resource. Argument type: ReleaseMode
`resourceName`	Name for the resource; must be unique among all resources. Argument type: String
`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. Argument type: String

Arguments	Descriptions
`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. Argument type: JobStatus
`stepName`	Takes the name of an existing step and creates a job step from this definition. It is used in conjunction with `procedureName` and `projectName` and defaults to the current job's procedure if neither are specified. If you use `stepName`, `createJobStep` uses the existing step as a template for the new dynamic `JobStep`. If you use this argument, you cannot use the `command` argument. You can use any project, procedure, or step as a template. If you do not provide a template, you must set the parameters of the new `JobStep` (such as `command`) manually.
`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. Argument type: String
`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. Argument type: String
`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. Argument type: String
`timeLimitUnits`	Specify `hours|minutes|seconds` for time limit units. Argument type: TimeLimitUnits

Arguments	Descriptions
workingDirectory	The ElectricFlow 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, ElectricFlow uses the directory it created for the job in the ElectricFlow workspace as the working directory. **Note:** If running a job step on a proxy resource, this directory must exist on the proxy target. Argument type: String
workspaceName	The name of the workspace where this job step log files will be stored. Argument type: String

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 ElectricFlow server using the
batch API.

# Create the batch API object
my $batch = $ec->newBatch('parallel');

# Create multiple requests
my @reqIds = (
    $batch->createJobStep(
        {
            parallel        => '1',
            projectName     => 'Default',
            subprocedure    => 'coolProcedure',
            actualParameter => [
                 {
                     actualParameterName => 'input',
                          value          => 'helloWorld'
                 }
             ],
         }
    ),
    $batch->createJobStep(
         {
             parallel        => '1',
             projectName     => 'Default',
             subprocedure    => 'coolProcedure',
             actualParameter => [
                  {
                      actualParameterName => 'input',
                            value         => 'helloWorld'
                  }
             ],
         }
     ),
);

# Send off the requests
$batch->submit();
```

ectool

syntax: `ectool createJobStep [optionals]`

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 that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID

Arguments	Descriptions
`status`	(Optional) `<pending|runnable|scheduled|running>` This 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. Argument type: JobStatus

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> [optionals]`

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 `jobStepId`.

Arguments	**Descriptions**
`jobStepId`	The unique ElectricFlow-generated identifier (a UUID) for a job step that is assigned automatically when the job step is created. The system also accepts a job step name assigned to the job step by its name template. Argument type: UUID
`actualParameter`	(Optional) 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. Argument type: Map
`alwaysRun`	(Optional) 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`. Argument type: Boolean
`broadcast`	(Optional) 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`. Argument type: Boolean
`command`	(Optional) The command to run. This argument is applicable to command job steps only. Argument type: String
`comment`	(Optional) The script to execute the functions in this step, which are passed to the step's shell for execution. Argument type: String
`condition`	(Optional) 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. Argument type: String

Arguments	Descriptions
`errorHandling`	(Optional) 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. Argument type: ErrorHandling
`exclusive`	(Optional) 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`". Argument type: Boolean
`exclusiveMode`	(Optional) 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. Argument type: ExclsiveMode
`logFileName`	(Optional) A custom log file name produced by running the job step. By default, ElectricFlow assigns a unique name for this file. Argument type: String

Arguments	Descriptions
`parallel`	(Optional) 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`. Argument type: Boolean
`postProcessor`	(Optional) 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. ElectricFlow 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). Argument type: String

Arguments	Descriptions
`precondition`	(Optional) 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 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. Argument type: String 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]`
`releaseExclusive`	(Optional) <*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`. Argument type: Boolean

Arguments	Descriptions
`releaseMode`	(Optional) 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. • `releaseToJob` - allows a job step to promote a "step exclusive" resource to a Job exclusive resource. Argument type: ReleaseMode
`resourceName`	(Optionals) Name for the resource; must be unique among all resources. Argument type: String
`shell`	(Optional) 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. Argument type: String
`status`	(Optional) `<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. Argument type: JobStatus
`subprocedure`	(Optional) 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. Argument type: String

Arguments	Descriptions
`subproject`	(Optional) 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. Argument type: String
`timeLimit`	(Optional) 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. Argument type: String
`timeLimitUnits`	(Optional) Specify `hours|minutes|seconds` for time limit units. Argument type: TimeLimitUnits
`workingDirectory`	(Optional) The ElectricFlow 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, ElectricFlow uses the directory it created for the job in the ElectricFlow workspace as the working directory. **Note:** If running a job step on a proxy resource, this directory must exist on the proxy target. Argument type: String
`workspaceName`	(Optional) The name of the workspace where this job step log files will be stored. Argument type: String

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> [optionals]`

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.

This command works only with ec-perl.

Arguments	Descriptions
`jobId`	The job to wait for. The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`timeout`	(Optional) The number of seconds to wait before giving up on a request. The default is 60 seconds. Argument type: Integer
`finalStatus`	(Optional) The status to wait for. Must be either "running" or "completed" (the default is "completed"). Argument type: String

Positional arguments

`jobId`

Response

Returns the result from the final `getJobStatus` query.

ec-perl

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

Examples

Wait for a job until it has a status of *completed*. This call would wait for 60 seconds, which is the default timeout:

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

Wait for a job until it has a status of *completed*. This call would wait for 30 seconds.:

```
$cmdr->waitForJob("4fa765dd-73f1-11e3-b67e-b0a420524153", 30);
```

Wait for a job until it has a status of *running*. This call would wait for 60 seconds, which is the default timeout:

```
$cmdr->waitForJob("4fa765dd-73f1-11e3-b67e-b0a420524153", undef, "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, go here.

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

Note: The `attachParameter` command in ElectricFlow 6.0 is not backward compatible with previous ElectricFlow releases.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument Type: String
`procedureName`	The name of the procedure to which the parameter is attached. Argument Type: String
`stepName`	The name of the procedure step to which the parameter is attached. Argument Type: String

Arguments	Descriptions
`formalParameterName`	The name of the parameter. Argument Type: String
`applicationName`	(Optional) The name of the application to which the parameter is attached. Argument Type: String
`componentApplicationName`	(Optional) The name of the component in the application to which the parameter is attached. Argument Type: String
`componentName`	(Optional) The name of the component to which the parameter is attached. Argument Type: String
`gateType`	(Optional) The type of the gate to which the parameter is attached: POST or PRE. Argument Type: GateType
`pipelineName`	(Optional) The name of the pipeline to which the parameter is attached. Argument Type: String
`processName`	(Optional) The name of the process to which the parameter is attached. Argument Type: String
`processStepName`	(Optional) The name of the process step to which the parameter is attached. Argument Type: String
`stageName`	(Optional) The name of the task to which the parameter is attached. Argument Type: String
`stateDefinitionName`	(Optional) The name of the workflow state definition to which the parameter is attached. Argument Type: String
`taskName`	(Optional) The name of the task to which the parameter is attached. Argument Type: String

Arguments	Descriptions
`workflowDefinitionName`	(Optional) The name of the workflow state definition to which the parameter is attached. Argument Type: String

Positional arguments

`projectName, procedureName, stepName, formalParameterName`

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->attachParameter("Default", "Daily Run", "Get Resources", "SCM Credential"
        {stageName => "PROD"}));
```

ectool

syntax: `ectool attachParameter <projectName> <procedureName> <stepName> <formalParameterName> [optionals]`

Example

```
ectool attachParameter "Default" "Daily Run" "Get Resources" "SCM Credential" --sta
geName "PROD"
```

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 runtime, 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
`projectName`	The name of the project containing the procedure. The project name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure containing the step. Argument type: String
`stepName`	The name of the step that calls a subprocedure. Argument type: String
`actualParameterName`	The name of the parameter. This name must be unique within the step, and at runtime, it must match the name of a formal parameter in the subprocedure. Argument type: String
`applicationName`	(Optional) The name of the application when the actual parameter is on an application process step. Argument type: String
`componentName`	(Optional) The name of the component when the actual parameter is on a component process step. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`processName`	(Optional) The name of the process when the actual parameter is on a process step. Argument type: String
`processStepName`	(Optional) The name of the process step when if the actual parameter is on a process step. Argument type: String
`releaseName`	(Optional) The name of the Release if the actual parameter is on a Release. Argument type: String
`scheduleName`	(Optional) The name of the schedule. Argument type: String

Arguments	Descriptions
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`value`	(Optional) This value is passed to the subprocedure as the value of the matching formal parameter. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String

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("Default", "Take Snapshot", "Retrieve", "Source direct
ory",
   {value => "local"});
```

ectool

syntax: `ectool <projectName> <procedureName> <stepName> <actualParameterName> [optionals]`

Example

```
ectool createActualParameter "Default" "Take Snapshot" "Retrieve" "Source director
y"
   --value "local"
```

Back to Top

createFormalParameter

Creates a formal parameter.

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

Arguments	Descriptions
`projectName`	The name of the project containing the procedure. The project name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure containing the parameter. Argument type: String
`formalParameterName`	The name for the formal parameter that is used when the procedure is invoked to specify a value for the parameter. Argument type: String
`applicationName`	(Optional) The name of the application when the actual parameter is on an application process step. Argument type: String
`componentName`	(Optional) The name of the component when the actual parameter is on a component process step. Argument type: String
`defaultValue`	(Optional) This value is used for the formal parameter when a value is not provided and the procedure is invoked. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`expansionDeferred`	(Optional) <*Boolean flag -* `0|1|true|false`> If `1` or `true`, expansion for this parameter is deferred. The parameter value is not expanded when the procedure call is expanded, but it can be expanded from a command step instead. The default is `false`, and the formal parameter is expanded immediately. Argument type: Boolean
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String

Arguments	Descriptions
`gateType`	(Optional) The type of the gate: POST or PRE. Argument Type: GateType
`label`	(Optional) The display label. Argument type: String
`orderIndex`	(Optional) The display order index starting at 1. Argument type: Integer
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`processName`	(Optional) The name of the process when the formal parameter is on a process. Argument type: String
`processStepName`	(Optional) The name of the process step when the formal parameter is on a process step. Argument type: String
`required`	(Optional) <*Boolean flag -* `0|1|true|false`> If 1 or true, this parameter is required. The procedure will not execute unless a value is given for it. Argument type: Boolean
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of a workflow state. Argument type: String
`taskName`	(Optional) The name of the task. Argument type: String
`type`	(Optional) The *type* can be any string value. This argument is used primarily by the web interface to represent custom form elements. When the type is the *credential* string value, the server expect a credential as the parameter value. Argument type: String

Arguments	Descriptions
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of a workflow. Argument type: String

Positional arguments

In ElectricFlow 5.0 and later, `projectName` and `formalParameterName`.

In releases earlier than ElectricFlow 5.0, `projectName`, `procedureName`, and `formalParameterName`.

For workflow state parameters: `projectName, formalParameterName, workflowDefinitionName` and `stateDefinitionName`

Response

Returns a `formalParameter` element.

ec-perl

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

For backward compatibility with releases earlier than ElectricFlow 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 a check box, radio button, and drop-down box

Check box 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 => '2',
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");
```

Drop-down menu example:

```
$ec->createFormalParameter(
$newProjectName,
"$buildprocedurename",
'BuildType',
{
type =>"select",
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");
```

ectool

For procedure parameters:

syntax: `ectool createFormalParameter <projectName> <formalParameterName> [optionals]`

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

syntax:`ectool createFormalParameter <projectName> <procedureName> <formalParameterName> [optionals]`

Example

```
ectool createFormalParameter "Sample Project" "Branch Name" --required 1
```

For workflow state parameters:

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

Example

```
ectool createFormalParameter --formalParameterName "Active users"
--projectName "Usage Report" --workflowDefinitionName "Usage
Workflow" --stateDefinitionName "Active and running"
```

Example using parameters to create a check box

You must create the `ec_customEditorData` property to add other parameters to the check box.

Back to Top

deleteActualParameter

Deletes an actual parameter.

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

Arguments	Descriptions
`projectName`	The name of the project that contains this actual parameter. The name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure that contains the step with this parameter. Argument type: String
`stepName`	The name of the step that contains the actual parameter. Argument type: String
`actualParameterName`	The name of the actual parameter to delete. Argument type: String
`applicationName`	(Optional) The name of the application when the actual parameter is on an application process step. Argument type: String

Arguments	Descriptions
`componentName`	(Optional) The name of the component when the actual parameter is on a component process step. Argument type: String
`processName`	(Optional) The name of the process when the actual parameter is on a process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the actual parameter is on a process step. Argument type: String
`releaseName`	(Optional) The name of the Release if the actual parameter is used in a Release. Argument type: String
`scheduleName`	(Optional) The name of the schedule containing the actual parameter. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String

Positional arguments

`projectName, procedureName, stepName, actualParameterName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteActualParameter('Default', 'CallSub', 'Take snapshot', 'Snapshot Versi
on');
```

ectool

syntax: `ectool deleteActualParameter <projectName> <procedureName> <stepName> <actualParameterName> [optionals]`

Example

```
ectool deleteActualParameter "Default" "CallSub" "Take snapshot" "Snapshot Version"
```

Back to Top

deleteFormalParameter

Deletes a formal parameter.

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

Arguments	Descriptions
`projectName`	The name of the project that contains the procedure or parameter to delete. The name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure that contains this parameter. Argument type: String
`formalParameterName`	The name of the formal parameter to delete. Argument type: String
`applicationName`	(Optional) The name of the application when the formal parameter is on an application process step. Argument type: String
`componentName`	(Optional) The name of the component when the formal parameter is on a component process step. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`gateType`	(Optional) The type of the gate: POST or PRE. Argument Type: GateType
`pipelineName`	(Optional) The name of the pipeline. Argument type: String

Arguments	Descriptions
`processName`	(Optional) The name of the process when the formal parameter is on a process. Argument type: String
`processStepName`	(Optional) The name of the process step when the formal parameter is on a process step. Argument type: String
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the workflow state. Argument type: String
`taskName`	(Optional) The name of the task. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of a workflow. Argument type: String

Positional arguments

In ElectricFlow 5.0 and later, `projectName` and `formalParameterName`.

In releases earlier than ElectricFlow 5.0, `projectName`, `procedureName`, and `formalParameterName`.

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteFormalParameter("Default", "Build Version", {pipelineName => "Q1 summa
ry"});
```

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

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

Example

```
$cmdr->deleteFormalParameter("Default", "QE testing", "Build Version");
```

ectool

syntax: `ectool deleteFormalParameter <projectName> <formalParameterName> [optionals]`

Example

```
ectool deleteFormalParameter "Default" "Build Version" --pipelineName "Q1 summary"
```

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

syntax:`ectool deleteFormalParameter <projectName> <procedureName> <formalParameterName> [optionals]`

Example

```
ectool deleteFormalParameter "Default" "QE testing" "Build Version"
```

Back to Top

detachParameter

Detaches a formal parameter from a step.

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

Arguments	Descriptions
`projectName`	The name of the project that contains this parameter. The name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure that contains this parameter. Argument type: String
`stepName`	The name of the step where this parameter is currently attached. Argument type: String
`formalParameterName`	The name of the formal parameter to detach. Argument type: String
`applicationName`	(Optional) The name of the application. Argument type: String

Arguments	Descriptions
`componentApplicationName`	(Optional) The name of the component in the application. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`gateType`	(Optional) The type of the gate. Argument type: GateType
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`processName`	The name of the process that contains this parameter. Argument type: String
`processStepName`	The name of the step in the process.. Argument type: String
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional)The name of the state definition. Argument type: String
`taskName`	(Optional) The name of the task. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String

Positional arguments

`projectName, procedureName, stepName, formalParameterName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool detachParameter <projectName> <procedureName> <stepName> <formalParameterName> [optionals]`

Example

```
ectool detachParameter "Default" "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 `projectName`, `procedureName`, and `stepName` arguments must be used together to specify the step.

Arguments	Descriptions
`actualParameterName`	The name of the actual parameter. Argument type: String
`applicationEntityRevisionId`	(Optional) The revision ID of the versioned object. Argument type: UUID
`applicationName`	(Optional)The name of the application when the actual parameter is on an application process step. The name must be unique among all projects. Argument type: String
`componentName`	(Optional)The name of the component when the actual parameter is on a component process step. Argument type: String
`flowName`	(Optional) Name of the flow that must be unique within the project. Argument Type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateId`	(Optional) The ID of the flow runtime state.. Argument Type: UUID
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String

Arguments	Descriptions
`flowStateName`	(Optional) Name of the flow state that must be unique within the flow. Argument Type: String
`flowTransitionName`	(Optional) Name of the flow transition that must be unique within the flow state. Argument Type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. You enter this argument to query a subprocedure call to the job step's parameter. Argument type: UUID
`procedureName`	(Optional) The name of the procedure when the actual parameter is on a procedure step. If you need actual parameters on a step, the `projectName`, `procedureName`, and `stepName` arguments must be used together to specify the step. Argument type: String
`processName`	(Optional) The name of the process when the actual parameter is on a process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the actual parameter is on a process step. Argument type: String
`projectName`	(Optional) The name of the project to query for the parameter on a schedule or procedure step. If you need actual parameters on a step, the `projectName`, `procedureName`, and `stepName` arguments must be used together to specify the step. Argument type: String
`releaseName`	(Optional) The name of the Release if the actual parameter is on it. Argument type: String

Arguments	Descriptions
`scheduleName`	(Optional) The name of the schedule to query for the parameter on a schedule. Argument type: String
`stateDefinitionName`	(Optional) The name of the workflow state definition. Argument type: String
`stateName`	(Optional) The name of the workflow state. Argument type: String
`stepName`	(Optional) The name of the step to query for the parameter on the step. Argument type: String
`transitionDefinitionName`	(Optional) The name of the workflow transition definition. Argument type: String
`transitionName`	(Optional) The name of the workflow transition. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String

Positional arguments

`actualParameterName`

Response

One `actualParameter` element.

ec-perl

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

Example

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

ectool

syntax: `ectool getActualParameter <actualParameterName> [optionals]`

Example

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

Back to Top

getActualParameters

Retrieves all actual parameters from a job, step, schedule, state, or transition. For more information about parameters, click here.

You must specify object locators to find the parameter. To find parameters on a step, you must use `projectName`, `procedureName`, and `stepName` to specify the step.

Arguments	Descriptions
`applicationEntityRevisionId`	(Optional) The revision ID of the versioned object. Argument type: UUID
`applicationName`	(Optional) The name of the application when the actual parameters are on an application process step. The name must be unique among all projects. Argument type: String
`componentName`	(Optional) The name of the component when the actual parameters are on a component process step. Argument type: String
`flowName`	(Optional) Name of the flow that must be unique within the project. Argument Type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateId`	(Optional) The ID of the flow runtime state. Argument Type: UUID
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) Name of the flow state that must be unique within the flow. Argument Type: String

Arguments	Descriptions
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`procedureName`	(Optional) The name of the procedure containing the parameters. To find parameters on a step, you must use `projectName`, `procedureName`, and `stepName` to specify the step. Argument type: String
`processName`	(Optional) The name of the process when the actual parameters are on a process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the actual parameters are on a process step. Argument type: String
`projectName`	(Optional) The name of the project containing the parameters. To find parameters on a step, you must use `projectName`, `procedureName`, and `stepName` to specify the step. Argument type: String
`releaseName`	(Optional) The name of the Release if the actual parameter is on it. Argument type: String
`scheduleName`	(Optional) The name of the schedule containing parameters. Argument type: String
`stateDefinitionName`	(Optional) The name of the workflow state definition. Argument type: String
`stateName`	(Optional) The name of the workflow state. Argument type: String

Arguments	Descriptions
`stepName`	(Optional) The name of the step containing parameters. To find parameters on a step, you must use `projectName`, `procedureName`, and `stepName` to specify the step. Argument type: String
`transitionDefinitionName`	(Optional) The name of the workflow transition definition. Argument type: String
`transitionName`	(Optional) The name of the workflow transition. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String

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 ({<optionals>});`

Example

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

ectool

syntax: `ectool getActualParameters [optionals]`

Example

```
ectool getActualParameters --projectName "Sample Project"
  --procedureName "CallSub" --stepName "Step1"
```

Back to Top

getFormalParameter

Retrieves a formal parameter by its name.

In ElectricFlow 5.0 and later, you must specify `projectName` and `formalParameterName`.

Arguments	Descriptions
`projectName`	The name of the project containing the procedure. The name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure containing the formal parameter. In ElectricFlow 5.0 and later, this argument is optional. Argument type: String
`formalParameterName`	The name for the formal parameter that is used when the procedure is invoked to specify a value for the parameter. Argument type: String
`applicationEntityRevisionId`	(Optional) The revision ID of the versioned object. Argument type: UUID
`applicationName`	(Optional) The name of the application when the formal parameter is on an application process step. Argument type: String
`componentName`	(Optional) The name of the component when the formal parameter is on a component process step. Argument type: String
`gateType`	(Optional) The type of the gate: POST or PRE. Argument Type: GateType
`pipelineName`	(Optional) The name of the pipeline when the formal parameter is on a pipeline. Argument type: String
`processName`	(Optional) The name of the process when the formal parameter is on a process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the formal parameter is on a process step. Argument type: String
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the workflow state definition. Argument type: String

Arguments	Descriptions
`stateName`	(Optional) The name of the workflow state. Argument type: String
`taskName`	(Optional) The name of the task. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String

Positional arguments

In ElectricFlow 5.0 and later, `projectName` and `formalParameterName`.

In releases earlier than ElectricFlow 5.0, `projectName`, `procedureName`, and `formalParameterName`.

Response

One `formalParameter` element.

ec-perl

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

Example

```
$cmdr->getFormalParameter("Default", "Test Case");
```

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

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

Example

```
$cmdr->getFormalParameter("Default","Get Sources", "Test Case");
```

ectool

syntax: `ectool getFormalParameter <projectName> <formalParameterName> [optionals]`

Example

```
ectool getFormalParameter "Default" "Test Case"
```

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

syntax: `ectool getFormalParameter <projectName> <procedureName> <formalParameterName> [optionals]`

Example

```
ectool getFormalParameter "Default" "Get Sources" "Test Case"
```

Back to Top

getFormalParameterOptions

Retrieves possible option values for a procedure's formal parameter in the procedure using the options script registered for it. The result may be used to dynamically populate a drop-down or any multi-select UI component.

You must specify `formalParameterName` and either `projectName` or `pluginName`.

Arguments	Descriptions
`formalParameterName`	The name of the parameter for which the options are requested. Argument type: String
`procedureName`	The name of the procedure containing the formal parameter. Argument type: String
`actualParameters`	(Optional) Parameters passed as arguments to the script. Argument type: String
`configurationParameters`	(Optional) Configuration parameter values passed as arguments to the script. Argument type: Map
`credentials`	(Optional) Credentials to be used in the script. Argument type: Collection
`pluginName`	(Optional) The name of a plugin. Argument type: String
`projectName`	(Optional) The name of the project containing the procedure. The name must be unique among all projects. Argument type: String

Positional arguments

`formalParameterName`, `procedureName`

Response

One `actualParameter` element.

ec-perl

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

Example

```
$cmdr->getFormalParameterOptions("Extra Parameter", "CallSub",
    {"projectName" => "Sample Project",
     "pluginName" => "EC-S3"});
```

ectool

syntax: `ectool getFormalParameterOptions <formalParameterName> <procedureName> [optionals]`

Example

```
getFormalParameterOptions "Extra Parameter" "CallSub" --projectName "Sample Projec
t"
 --pluginName "EC-S3"
```

Back to Top

getFormalParameters

Retrieves all formal parameters from a procedure, schedule, step, or state definition.

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
`projectName`	The name of the project containing the parameters. The name must be unique among all projects. Argument type: String
`applicationEntityRevisionId`	(Optional) The revision ID of the versioned object. Argument type: UUID
`applicationName`	(Optional) The name of the application when the formal parameters are on an application process step. Argument type: String
`componentName`	(Optional) The name of the component when the formal parameters are on a component process step. Argument type: String
`gateType`	(Optional) The type of the gate: POST or PRE. Argument Type: GateType
`pipelineName`	(Optional) The name of the pipeline when the formal parameter is on a pipeline. Argument type: String

Arguments	Descriptions
`procedureName`	(Optional) The name of the procedure when the formal parameters are on a procedure. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) The name of the process when the formal parameters are on a process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the formal parameter is on a process step. Argument type: String
`scheduleName`	(Optional) The name of the schedule. When using this argument, you must also enter `projectName`. Argument type: String
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the workflow state definition. Argument type: String
`stateName`	(Optional) The name of the workflow state. Argument type: String
`stepName`	(Optional) The name of the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`taskName`	(Optional) The name of the task. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String

Positional arguments

`projectName` and 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("QA Test", {procedureName => "Build"});
```

ectool

syntax: `ectool getFormalParameters <projectName> [optionals]`

Example

```
getFormalParameters "QA 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.

In releases earlier than ElectricFlow 5.0, you must enter `projectName`, `procedureName`, and `actualParameterName` to modify procedure parameters.

In ElectricFlow 5.0 and later, you must enter `projectName, procedureName`, `stepName`, and `actualParameterName` to modify procedure parameters.

You must enter `projectName, actualParameterName, workflowDefinitionName` and `stateDefinitionName` for workflow state parameters.

Arguments	Descriptions
`projectName`	The name of the project containing this parameter. The name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure containing the step with this parameter. Argument type: String
`stepName`	The name of the step containing this parameter. Argument type: String
`actualParameterName`	The name of the actual parameter to modify. Argument type: String

Arguments	Descriptions
`applicationName`	(Optional) The name of the application when the actual parameter is on an application process step. Argument type: String
`componentName`	(Optional) The name of the component when the actual parameter is on a component process step. Argument type: String
`newName`	(Optional) New name of the parameter. Argument type: String
`processName`	(Optional) The name of the process when the actual parameter is on a process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the actual parameter is on a process step. Argument type: String
`releaseName`	(Optional) The name of the release if the actual parameter is on a Release. Argument type: String
`scheduleName`	(Optional) The name of the schedule. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`value`	(Optional) Changes the current value on an actual parameter. This value is passed to the subprocedure as the value of the matching formal parameter. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String

Response

Returns a modified actual parameter element.

ec-perl

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

Example

```
$cmdr->modifyActualParameter("Default", "Final run", "Take snapshot", "Software lev
el",
    {newName => "Software version"});
```

ectool

syntax: `ectool modifyActualParameter <projectName> <procedureName> <stepName> <actualParameterName> [optionals]`

Example

```
ectool modifyActualParameter "Default" "Final run" "Take snapshot" "Software level"
   --newName "Software version"
```

Back to Top

modifyFormalParameter

Modifies an existing formal parameter.

In releases earlier than ElectricFlow 5.0, you must enter `projectName`, `procedureName`, and `formalParameterName` to modify procedure parameters.

In ElectricFlow 5.0 and later, you must enter `projectName` and `formalParameterName` to modify procedure parameters.

You must enter `projectName, formalParameterName, workflowDefinitionName` and `stateDefinitionName` for workflow state parameters.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`formalParameterName`	The name for this formal parameter. It is used when the procedure is invoked to specify a value for the parameter. Argument type: String
`applicationName`	(Optional) The name of the application when the formal parameters are on an application process step. Argument type: String
`componentName`	(Optional) The name of the component when the formal parameters are on a component process step. Argument type: String

Arguments	Descriptions
`defaultValue`	(Optional) This value is used for the formal parameter if one is not provided when the procedure is invoked. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`expansionDeferred`	(Optional) <*Boolean flag -* `0|1|true|false`> If this is set to `1` or `true`, expansion for this parameter is deferred. The parameter value is not expanded when the procedure call is expanded, but it can be expanded from a command step instead. The default is `false`, and the formal parameter is expanded immediately. Argument type: Boolean
`flowName`	(Optional) The name of the flow. Argument type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`gateType`	(Optional) The type of the gate: POST or PRE. Argument Type: GateType
`label`	(Optional) The display label. Argument type: String
`newName`	(Optional) New name for the parameter. Argument type: String
`orderIndex`	(Optional) The display order index starting at 1. Argument type: Integer
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`procedureName`	(Optional) The name of the procedure containing this parameter. Argument type: String

Arguments	Descriptions
`processName`	(Optional) The name of the process when the formal parameter is on a process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the formal parameter is on a process step. Argument type: String
`required`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, this parameter is required. The procedure will not execute unless a value is given for it. Argument type: Boolean
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of a workflow state. Argument type: String
`taskName`	(Optional) The name of the task. Argument type: String
`type`	(Optional) The *type* can be any string value. This argument is used primarily by the web interface to represent custom form elements. When the type is the *credential* string value, the server expect a credential as the parameter value. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of a workflow. Argument type: String

Positional arguments

In ElectricFlow 5.0 and later, for procedure parameters: `projectName` and `formalParameterName`.

In releases earlier than ElectricFlow 5.0, for procedure parameters: `projectName`, `procedureName`, and `formalParameterName`.

For workflow state parameters: `projectName, formalParameterName, workflowDefinitionName` and `stateDefinitionName`.

Response

Returns a modified formal parameter element.

ec-perl

For procedural parameters in ElectricFlow 5.0 and later:

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

Example

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

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

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

ectool

For procedural parameters in ElectricFlow 5.0 and later:

syntax: `ectool modifyFormalParameter <projectName> <formalParameterName> [optionals]`

Example

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

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

syntax: `ectool modifyFormalParameter <projectName> <procedureName> <formalParameterName> [optionals]`

For workflow state parameters:

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

Back to Top

validateFormalParameters

Validates input parameters for a procedure using the validation script registered for it.

You must enter the `procedureName`.

Arguments	Descriptions
`procedureName`	The name of a procedure. Argument type: String
`actualParameters`	(Optional) Parameters passed as arguments to the script.. Argument type: String

Arguments	Descriptions
`configurationParameters`	(Optional) Configuration parameter values passed as arguments to the script. Argument type: Map
`credentials`	(Optional) Credentials to be used in the script. Argument type: Collection
`pluginName`	(Optional) The name of a plugin. Argument type: String
`projectName`	(Optional) The name of a project. Argument type: String

Positional arguments

`projectName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->validateFormalParameters("Shopping Cart", {projectName => "Default"});
```

ectool

syntax: `ectool validateFormalParameters <procedureName> [optionals]`

Example

```
validateFormalParameters "Shopping Cart" --projectName "Default"
```

Back to Top

API Commands - Pipelines and Releases

abortAllPipelineRuns

Aborts all pipeline runs associated with a release.

Arguments	Descriptions
`force`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the pipelines abort immediately. A zero value allows the running tasks in the pipeline stage to terminate. Argument Type: Boolean
`pipelineName`	(Optional) The name of the pipeline. Argument Type: String
`projectName`	The name of the project. This must be unique among all projects. Argument Type: String
`releaseName`	(Optional) The name of the release. Argument Type: String

Positional arguments

`projectName`

Response

None or a status OK message.

ec-perl

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

Example

```
$ec->abortAllPipelineRuns("Production", {force => true, pipelineName => "bankin
g"});
```

ectool

syntax: `ectool abortAllPipelineRuns <projectName> [optionals]`

Example

```
ectool abortAllPipelineRuns "Production" --force true --pipelineName "banking"
```

Back to Top

abortPipelineRun

Aborts a pipeline run.

Arguments	Descriptions
`flowRuntimeId`	(Optional) The ID of the flow runtime. Argument Type: UUID
`flowRuntimeName`	(Optional) The name of the flow runtime. Argument Type: String
`force`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the pipeline aborts immediately. A zero value allows the running task in the pipeline stage to terminate. Argument Type: Boolean
`projectName`	(Optional) The name of the project. Argument Type: String

Positional arguments

None

Response

None or a status OK message.

ec-perl

syntax: `$<object>->abortPipelineRun( {<optionals>});`

Example

```
$ec->abortPipelineRun({flowRuntimeName => "Final Release", projectName => "Producti
on"});
```

ectool

syntax: `ectool abortPipelineRun [optionals]`

Example

```
ectool abortPipelineRun --flowRuntimeName "Final Release" --projectName "Productio
n"
```

Back to Top

completeManualTask

Completes the manual task.

You must specify the `flowRuntimeId`, `stageName`, and `taskName` arguments.

Arguments	Descriptions
`flowRuntimeId`	ID of the flow runtime. Argument Type: UUID
`stageName`	Name of the stage. Argument Type: String
`taskName`	Name of the task. Argument Type: String
`action`	(Optional) The type of action to be taken on the pipeline stage manual task. Valid values are `completed` and `failed`. Argument type: ManualProcessStepStatus
`actualParameters`	(Optional) Specifies the list of values to pass as parameters to the flow. 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. Argument Type: Map
`evidence`	(Optional) User-supplied evidence provided while performing an action on the pipeline stage manual task or stage gate task. Argument type: String
`gateType`	(Optional) The type of the gate. Argument type: GateType

Positional arguments

`flowRuntimeId`, `stageName`, `taskName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->completeManualTask(<flowRuntimeId>, <stageName>, <taskName>, {<optionals>});`

Example

```
$ec->completeManualTask("4fa765dd-73f1-11e3-b67e-b0a420524153", "Final Deploy", "Co
nfirm version"});
```

ectool

syntax: `ectool completeManualTask <flowRuntimeId> <stageName> <taskName> [optionals]`

Example

```
ectool completeManualTask 4fa765dd-73f1-11e3-b67e-b0a420524153 "Final Deploy" "Conf
irm version"
```

Back to Top

completeRelease

Completes the Release.

Arguments	Descriptions
`projectName`	(Optional) The name of the project. Argument Type: Map
`releaseId`	(Optional) The release ID. Argument Type: UUID
`releaseName`	(Optional) The name of the release. Argument Type: String

Response

None or a status OK message.

ec-perl

syntax: `$<object>->completeRelease( {<optionals>});`

Example

```
$ec->completeRelease({projectName => "Production", releaseName => "Final"});
```

ectool

syntax: `ectool completeRelease [optionals]`

Example

```
ectool completeRelease --projectName "Production" --releaseName "Final"
```

Back to Top

createDeployer

Creates a new Deployer task for a project or a Release.

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

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument Type: String
`deployerName`	The name of the Deployer task that must be unique within the project. Argument Type: String

Positional arguments

`projectName`, `deployerName`

Response

Returns a Deployer object.

ec-perl

syntax: `$<object>->createDeployer(<projectName>, <deployerName>);`

Example

```
$ec->createDeployer("Default", "Deploy Shopping Cart");
```

ectool

syntax: `ectool createDeployer <projectName> <deployerName>`

Example

```
ectool createDeployer "Default" "Deploy Shopping Cart"
```

Back to Top

createDeployerApplication

Adds a Deployer application to a Release.

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

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument Type: String
`applicationName`	The name of the application. Argument Type: String
`applicationProjectName`	(Optional) The name of the project containing the application. If this argument is not specified, the default is the Release project name. Argument Type: String
`errorHandling`	(Optional) The error handling method for the Deployer application. Argument Type: FlowStateErrorHandling
`orderIndex`	(Optional) The order in which the applications are deployed, starting from 1. Argument Type: String
`processName`	(Optional) The name of the application process. Argument Type: String
`releaseName`	(Optional) The name of the Release. Argument Type: String
`smartDeploy`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the pipeline uses smart deploy when the application is deployed. immediately. ElectricFlow deploys the application only with artifacts that have not been deployed to a resource or with selected versions of the artifact have not been deployed to new resources since a previous run. When this argument is set to `false` or `0`, the pipeline deploys all the application processes, components, and artifacts in the application, referred to as a *full run*. Argument Type: Boolean
`snapshotName`	(Optional) The name of the application snapshot. Argument Type: String

Arguments	Descriptions
`stageArtifacts`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, artifact staging is enabled for the application process. ElectricFlow retrieves artifacts that will be deployed in an application run before the deployment starts. When this argument is set to `false` or `0`, artifact staging is disabled. Argument Type: Boolean

Positional arguments

`projectName`, `applicationName`

Response

Returns a Deployer application object.

ec-perl

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

Example

```
$ec->createDeployerApplication("Default", "Shopping Cart", {applicationProjectName
=> "Deploy cart"});
```

ectool

syntax: `ectool createDeployerApplication <projectName> <applicationName> [optionals]`

Example

```
ectool createDeployerApplication "Default" "Shopping Cart" --applicationProjectName
"Deploy cart"
```

Back to Top

createDeployerConfiguration

Adds the Deployer configuration to the Deployer application.

You must specify the `projectName`, `applicationName`, and `stageName`.

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

Arguments	Descriptions
`stageName`	Name of the stage in the pipeline. If a Release is also specified, the pipeline is defined in the Release. Argument Type: String
`actualParameters`	(Optional) Specifies the list of values to pass as parameters to the flow. 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. Argument Type: Map
`applicationProjectName`	(Optional) The name of the project containing the application. If this argument is not specified, the default is the Release project name. Argument Type: String
`environmentName`	(Optional) Name of the environment. Argument Type: String
`environmentProjectName`	(Optional) Name of the project containing the specified environment or environment template. If this argument is not specified, the default is the Release project name. Argument Type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument Type: String
`environmentTemplateProjectName`	(Optional) Name of the project containing specified environment template. If this argument is not specified, the default is the environment project name. Argument Type: String
`insertRollingDeployManualStep`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` a manual step needs to be added after each phase or batch is run. Argument type: Boolean
`releaseName`	(Optional) The name of the Release. Argument Type: String
`rollingDeployEnabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` the pipeline runs the rolling deployment. Argument type: Boolean

Arguments	Descriptions
`rollingDeployManualStepAssignees`	(Optional) A list of assignees who receive the notification when the rolling deploy iteration is completed. Argument type: Collection
`rollingDeployManualStepCondition`	(Optional) The **Run if** condition (run condition) on the manual step that was created during the rolling deployment. Argument type: NotificationType
`rollingDeployPhases`	(Optional) One or more rolling deploy phases to be used in the rolling deployment. Argument type: Collection
`skipDeploy`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the application is not deployed to an environment. Argument type: Boolean

Positional arguments

`projectName`, `applicationName`, `stageName`

Response

Returns a Deployer configuration object.

ec-perl

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

Example

```
$ec->createDeployerConfiguration("Default", "Shopping Cart", "PROD", {releaseName =
> "Q1 Summary"});
```

ectool

syntax: `ectool createDeployerConfiguration <projectName> <applicationName> <stageName> [optionals]`

Example

```
ectool createDeployerConfiguration "Default" "Shopping Cart" "PROD" --releaseName "
Q1 Summary"
```

Back to Top

createGate

Creates a new gate for a stage.

You must specify the `projectName`, `stageName`, and `gateType`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`stageName`	Name of the stage. Argument Type: String
`gateType`	Type of the gate. Valid values are `PRE` and `POST`. Argument Type: GateType
`condition`	(Optional) The **Run if** condition (run condition) is property reference that the pipeline evaluates before executing the next stage. The pipeline waits until one or more dependent run conditions are met. By default, no run conditions are set. The run condition is "fixed text" or text embedding property references that are evaluated into a logical `TRUE` or `FALSE`. An \"0\" or \"false\" is interpreted as `FALSE`. An empty string or any other result is interpreted as `TRUE`. The property reference can be a JavaScript expression, for example, this expression could test whether the name of a step is equal to the value of a property called "restartStep". `$[/javascript (myStep.stepName == myJob.restartStep)]` Argument Type: String
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String
`precondition`	(Optional) A property that allows the stage to be created with "pause", which then pauses the pipeline at that point. The pipeline waits until one or more dependent conditions are met. When the pipeline status is eligible to transition from pending to runnable, a **Wait until** condition (precondition) is evaluated. The precondition is fixed text or text embedding a 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 pipeline does not progress until the condition is `TRUE`. By default, this condition is not set and is `FALSE`.. Argument type: String

Positional arguments

`projectName`, `stageName`, `gateType`

Response

Returns a pipeline gate object.

ec-perl

syntax: `$<object>->createGate(<projectName>, <stageName>, <gateType>, {<optionals>});`

Example

```
$ec->createGate("Default", "Preflight", POST, {pipelineName => "Quarterly Summar
y"});
```

ectool

syntax: `ectool createGate <projectName> <stageName> <gateType> [optionals]`

Example

```
ectool createGate "Default" "Preflight" POST --pipelineName "Quarterly Summary"
```

Back to Top

createNote

Creates a new note.

You must specify `note`.

Arguments	Descriptions
`note`	Notes about the entity. Argument Type: String
`noteName`	(Optional) Name of the note. Argument Type: String
`projectName`	(Optional) Project name of the entity that owns the note. Argument Type: String
`releaseName`	(Optional) The name of the Release that references the pipeline. Argument Type: String

Positional arguments

`note`

Response

Returns a Deployer application object.

ec-perl

syntax: `$<object>->createNote(<note>, {<optionals>});`

Example

```
$ec->createNote("Approved by ABC", {releaseName => "Production"});
```

ectool

syntax: `ectool createNote <note> [optionals]`

Example

```
ectool createNote "Approved by ABC" --releaseName "Production"
```

Back to Top

createPipeline

Creates a new pipeline for a project.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`pipelineName`	Name of the pipeline that must be unique among all projects. Argument Type: String
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
`enabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the pipeline is enabled. Argument type: Boolean
`type`	(Optional) Type of pipeline. Argument Type: PipelineType

Positional arguments

`projectName`, `pipelineName`

Response

Returns a pipeline object.

ec-perl

syntax: `$<object>->createPipeline(<projectName>, <pipelineName>, {<optionals>});`

Example

```
$ec->createPipeline("Default", "Web Server Image", {description => "Amazon"});
```

ectool

syntax: `ectool createPipeline <projectName> <pipelineName> [optionals]`

Example

```
ectool createPipeline "Default" "Web Server Image" --description "Amazon"
```

Back to Top

createRelease

Creates a new Release for a project.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`releaseName`	Name of the Release. Argument Type: String
`actualParameters`	(Optional) Actual parameters. Argument Type: Map
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
`pipelineName`	(Optional) Name of the pipeline. Argument Type: String
`pipelineProjectName`	(Optional) The name of the project containing specified pipeline. If this argument is not specified, the default is the release project name. Argument Type: String
`plannedEndDate`	(Optional) The date when this release is expected to end (for example, 2016-05-15). Argument Type: String

Arguments	Descriptions
plannedStartDate	(Optional) The date when this release is expected to begin (for example, 2016-05-15) Argument Type: String

Positional arguments

projectName, releaseName

Response

Returns a Release object.

ec-perl

syntax:`$<object>->createRelease(<projectName>, <releaseName>, {<optionals>});`

Example

```
$ec->createRelease("Default", "Production", {description => "Pet Shop Web Site", pi
pelineName => "Daily Update"});
```

ectool

syntax:`ectool createRelease <projectName> <releaseName> [optionals]`

Example

```
ectool createRelease "Default" "Production" --description "Pet Shop Web Site" --pip
elineName "Daily Update"
```

Back to Top

createStage

Creates a new stage in a pipeline for a project.

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

Arguments	Descriptions
projectName	Name of the project that must be unique among all projects. Argument Type: String
stageName	Name of the stage. Argument Type: String
afterStage	(Optional) The stage that is placed after the new stage. Argument Type: String

Arguments	Descriptions
`beforeStage`	(Optional) The stage that is placed before the new stage. Argument Type: String
`condition`	(Optional) The **Run if** condition (run condition) is property reference that the pipeline evaluates before executing the next stage. The pipeline waits until one or more dependent run conditions are met. By default, no run conditions are set. The run condition is "fixed text" or text embedding property references that are evaluated into a logical `TRUE` or `FALSE`. An \"0\" or \"false\" is interpreted as `FALSE`. An empty string or any other result is interpreted as `TRUE`. The property reference can be a JavaScript expression, for example, this expression could test whether the name of a step is equal to the value of a property called "restartStep". `$[/javascript (myStep.stepName == myJob.restartStep)]` Argument Type: String
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String
`precondition`	(Optional) A property that allows the stage to be created with "pause", which then pauses the pipeline at that point. The pipeline waits until one or more dependent conditions are met. When the pipeline status is eligible to transition from pending to runnable, a **Wait until** condition (precondition) is evaluated. The precondition is fixed text or text embedding a 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 pipeline does not progress until the condition is `TRUE`. By default, this condition is not set and is `FALSE`.. Argument type: String

Positional arguments

`projectName`, `stageName`

Response

Returns a pipeline stage object.

ec-perl

syntax: `$<object>->createStage(<projectName>, <stageName>, {<optionals>});`

Example

To specify that the Preflight stage run after the "Automated Tests" stage, enter:

```
$ec->createStage("Default", "Preflight", {afterStage => "Automated Tests"});
```

To set the **Wait until** condition (precondition) that the "Automated Tests" stage is completed before running the "Preflight" stage, enter:

```
$ec->createStage("Default", "Preflight", {precondition => "$[/myPipelineRuntime/aut
oTestCompleted]"});
```

To set the **Run if** condition (run condition) that the name of the next stage is "Preflight", enter:

```
$ec->createStage("Default", "Preflight", {condition => "$[/javascript (myPipelineRu
ntime.pipelineName == myPipelineRuntime.Preflight)]"});
```

ectool

syntax: `ectool createStage <projectName> <stageName> [optionals]`

Example

To specify that the Preflight stage run after the "Automated Tests" stage, enter:

```
ectool createStage "Default" "Preflight" --afterStage "Automated Test"
```

To set the **Wait until** condition (precondition) that the "Automated Tests" stage is completed before running the "Preflight" stage, enter:

```
ectool createStage "Default" "Preflight" --precondition "$[/myPipelineRuntime/autoT
estCompleted]"
```

To set the **Run if** condition (run condition) that the name of the next stage is "Preflight", enter:

```
ectool createStage "Default" "Preflight" --condition "$[/javascript (myPipelineRunt
ime.pipelineName == myPipeline.Preflight)]"
```

Back to Top

createTask

Creates a new task for a task container.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`taskName`	Name of the task. Argument Type: String

Arguments	Descriptions
`actualParameters`	(Optional) Specifies the list of values to pass as parameters to the flow. 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. Argument Type: Map
`advancedMode`	(Optional) <*Boolean flag* - `0|1|true|false`> When `advancedMode` is set to `true` or `1`, advanced mode is enabled so that all validations for the `applicationName`, `environmentName`, `processName`, or `snapshotName` will be skipped. These objects can be represented using the "`$[]`" notation and later expanded at the pipeline runtime. For example, when `advancedMode` is enabled, the system does not validate that the environment exists or whether it is mapped to an application. Argument type: Boolean
`afterTask`	(Optional) The task that is placed after the named task. Argument Type: String
`approvers`	(Optional) A list of task approvers who receive the notification. Argument Type: Collection
`beforeTask`	(Optional) The task that is placed before the named task. Argument Type: String
`condition`	(Optional) The **Run if** condition (run condition) is property reference that the pipeline evaluates before executing the next task. The pipeline waits until one or more dependent run conditions are met. By default, no run conditions are set. The run condition is "fixed text" or text embedding property references that are evaluated into a logical `TRUE` or `FALSE`. An \"0\" or \"false\" is interpreted as `FALSE`. An empty string or any other result is interpreted as `TRUE`. The property reference can be a JavaScript expression, for example, this expression could test whether the name of a step is equal to the value of a property called "restartStep". `$[/javascript (myStep.stepName == myJob.restartStep)]` Argument Type: String
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String

Arguments	Descriptions
`enabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the task is enabled. Argument type: Boolean
`environmentName`	(Optional) Name of the environment to create from a template. When the `environmentTemplateName` is specified, the `environmentName` can be specified using the "`$[]`" notation. It is expanded at runtime. Argument Type: String
`environmentProjectName`	(Optional) Name of the project to which the environment or environment template belongs. Argument Type: String
`environmentTemplateName`	(Optional) Name of the environment template to use. When the `environmentTemplateName` is specified, the `environmentName` can be specified using the "`$[]`" notation. It is expanded at runtime. Argument Type: String
`environmentTemplateProjectName`	(Optional) Name of the project containing specified environment template. If this argument is not specified, the default is the environment project name. Argument Type: String
`errorHandling`	(Optional) Type of error handling for this task. Argument Type: FlowStateErrorHandling
`firstTask`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, this task is the first task in the stage, and the `beforeTask` and `afterTask` arguments should not be used. The default is `false` or `0`. Argument type: Boolean
`gateType`	(Optional) The type of the gate. Argument Type: GateType
`insertRollingDeployManualStep`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` a manual step needs to be added after each phase or batch is run. Argument type: Boolean

Arguments	Descriptions
`instruction`	(Optional) Instruction associated with the task. Argument Type: String
`keepOnError`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the system is set to keep environment if an error occurs. The default is `false` or `0`. Argument type: Boolean
`notificationTemplate`	(Optional) String containing email formatting instructions for generating notifications. Argument type: String
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String
`precondition`	(Optional) A property that allows the task to be created with "pause", which then pauses the pipeline at that point. The pipeline waits until one or more dependent conditions are met. When the pipeline status is eligible to transition from pending to runnable, a **Wait until** condition (precondition) is evaluated. The precondition is fixed text or text embedding a 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 pipeline does not progress until the condition is `TRUE`. By default, this condition is not set and is `FALSE`.. Argument type: String
`skippable`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the task can be skipped in the pipeline. Argument type: Boolean
`rollingDeployEnabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` the pipeline runs the rolling deployment. Argument type: Boolean
`rollingDeployManualStepAssignees`	(Optional) A list of assignees who receive the notification when the rolling deploy iteration is completed. Argument type: Collection

Arguments	Descriptions
rollingDeployManualStepCondition	(Optional) The **Run if** condition (run condition) on the manual step that was created during the rolling deployment. Argument type: NotificationType
rollingDeployPhases	(Optional) One or more rolling deploy phases to be used in the rolling deployment. Argument type: Collection
snapshotName	(Optional) Name of the snapshot associated with the application. When `advancedMode` is set to `true` or `1`, advanced mode is enabled so that all validations for the `applicationName`, `environmentName`, `processName`, or `snapshotName` will be skipped. These objects can be represented using the "`$[]`" notation and later expanded at the pipeline runtime. For example, when `advancedMode` is enabled, the system does not validate that the environment exists or whether it is mapped to an application. Argument type: String
stageName	(Optional) Name of the stage to which the task belongs. Argument Type: String
startTime	(Optional) The time to begin invoking this task. The time is formatted `hh:mm` using the 24-hours clock (for example, 17:00). Argument Type: String
subapplication	(Optional) Name of the application that owns the subprocess. Argument Type: String
subpluginKey	(Optional) Name of the pluginKey for a procedure when the procedure is referenced. Argument Type: String
subprocedure	(Optional) Name of the subprocedure when a procedure is referenced. Argument Type: String
subprocess	(Optional) Name of the ElectricFlow process. Argument Type: String
subproject	(Optional) Name of the project in which the procedure runs. Argument Type: String

Arguments	Descriptions
`subworkflowDefinition`	(Optional) Name of the workflow definition in which the workflow definition is referenced. Argument Type: String
`subworkflowStartingState`	(Optional) Name of the starting state in the specified workflow definition. Argument Type: String
`taskProcessType`	(Optional) The type of the process that the task can invoke. Argument Type: TaskProcessType
`taskType`	(Optional) The type of the task. Argument Type: TaskType
`tierResourceCounts`	(Optional) The resource count per resource template tier. Argument Type: Map
`workspaceName`	(Optional) The name of the workspace. Argument Type: String

Positional arguments

`projectName`, `taskName`

Response

Returns a task object.

ec-perl

syntax: `$<object>->createTask(<projectName>, <taskName>, {<optionals>});`

Example

To run the "Save results" task before the "Save log file" task:

```
$ec->createTask("Default", "Save results", {beforeTask => "Save log file"});
```

To create a task as the first task in a stage:

```
$ec->createTask("Default", "Enter user name and password", {firstTask => true});
```

To create a task with **Run if** and **Wait until** conditions:

```
$ec->createTask("Default", "Deploy WAR file", {condition => $[/javascript (myTask.t
askName == myTask.deployWar), precondition => $[/myTask/autoTestCompleted]]});
```

ectool

syntax: `ectool createTask <projectName> <taskName> [optionals]`

Example

To run the "Save results" task before the "Save log file" task:

```
ectool createTask "Default" "Save results" --beforeTask "Save log file"
```

To create a task as the first task in a stage:

```
ectool createTask "Default" "Enter user name and password" --firstTask true
```

To create a task with **Run if** and **Wait until** conditions:

```
ectool createTask "Default" "Deploy WAR file" --condition "$[/javascript (myTask.ta
skName == myTask.deployWar)" --precondition "$[/myTask/autoTestCompleted]"
```

Back to Top

deleteGate

Deletes a gate in a stage.

You must specify the `projectName`, `stageName`, and `gateType`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`stageName`	Name of the stage. Argument Type: String
`gateType`	Type of the gate. Valid values are `PRE` and `POST`. Argument Type: GateType
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String

Positional arguments

`projectName`, `stageName`, `gateType`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteGate(<projectName>, <stageName>, <gateType>, {<optionals>});`

Example

```
$ec->deleteGate("Default", "Preflight", POST, {pipelineName => "Quarterly Summar
y"});
```

ectool

syntax: `ectool deleteGate <projectName> <stageName> <gateType> [optionals]`

Example

```
ectool deleteGate "Default" "Preflight" POST --pipelineName "Quarterly Summary"
```

Back to Top

deleteNote

Deletes a note associated with an entity.

You must specify `note`.

Arguments	Descriptions
`note`	Notes provided about the entity. Argument Type: String
`projectName`	(Optional) Project name of the entity that owns the note. Argument Type: String
`releaseName`	(Optional) The name of the Release. Argument Type: String

Positional arguments

`note`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteNote(<note>, {<optionals>});`

Example

```
$ec->deleteNote("Approved by ABC", {releaseName => "Production"});
```

ectool

syntax: `ectool deleteNote <note> [optionals]`

Example

```
ectool deleteNote "Approved by ABC" --releaseName "Production"
```

Back to Top

deletePipeline

Deletes a pipeline in a project.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`pipelineName`	Name of the pipeline that must be unique among all projects. Argument Type: String

Positional arguments

`projectName, pipelineName`

Response

None or a status OK message.

ec-perl

syntax:`$<object>->deletePipeline(<projectName>, <pipelineName>);`

Example

```
$ec->deletePipeline("Default", "Web Server Image");
```

ectool

syntax:`ectool deletePipeline <projectName> <pipelineName>`

Example

```
ectool deletePipeline "Default" "Web Server Image"
```

Back to Top

deletePipelineRun

Deletes a pipeline runtime.

Arguments	Descriptions
`flowRuntimeId`	(Optional) The ID of the flow runtime. Argument Type: UUID
`flowRuntimeName`	(Optional) The name of the flow runtime. Argument Type: String
`projectName`	(Optional) The name of the project. Argument Type: String

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deletePipelineRun({<optionals>});`

Example

```
$ec->deletePipelineRun({flowRuntimeName => "Test Run", projectName => "Default");
```

ectool

syntax: `ectool deletePipelineRun [optionals]`

Example

```
ectool deletePipelineRun --flowRuntimeName "Test Run" --projectName "Default"
```

Back to Top

deleteRelease

Deletes a Release.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`releaseName`	Name of the Release. Argument Type: String

Positional arguments

`projectName`, `releaseName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteRelease(<projectName>, <releaseName>);`

Example

```
$ec->deleteRelease("Default", "Pet Store");
```

ectool

syntax: `ectool deleteRelease <projectName> <releaseName>`

Example

```
ectool deleteRelease "Default" "Pet Store"
```

Back to Top

deleteStage

Deletes a stage in a project.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`stageName`	Name of the stage. Argument Type: String
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument Type: String

Positional arguments

`projectName`, `stageName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteStage(<projectName>, <stageName>, {<optionals>});`

Example

```
$ec->deleteStage("Default", "PROD", {pipelineName => "Q2 Summary"});
```

ectool

syntax: `ectool deleteStage <projectName> <stageName> [optionals]`

Example

```
ectool deleteStage "Default" "PROD" --pipelineName "Q2 Summary"
```

Back to Top

deleteTask

Deletes a task in a task container.

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

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

Arguments	Descriptions
`taskName`	Name of the task. Argument Type: String
`gateType`	(Optional) The type of the gate. Argument Type: GateType
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String
`stageName`	(Optional) Name of the stage to which the task belongs. Argument Type: String

Positional arguments

`projectName`, `taskName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteTask(<projectName>, <taskName>, {<optionals>});`

Example

```
$ec->deleteTask("Default", "Save image", {pipelineName => "Production"});
```

ectool

syntax: `ectool deleteTask <projectName> <taskName> [optionals]`

Example

```
ectool deleteTask "Default" "Save image" --pipelineName "Production"
```

Back to Top

getAllWaitingTasks

Retrieves a list of all tasks across pipeline runs that are awaiting manual approval.

Arguments	Descriptions
`projectName`	Name of the project. The name must be unique among all projects. Argument type: String
`releaseName`	(Optional) Name of the release. Argument type: String

Positional arguments

`projectName`

Response

Returns all waiting task objects across pipeline runs that are awaiting manual approval.

ec-perl

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

Example

```
$ec->getAllWaitingTasks("proj1", {releaseName => "release1"});
```

ectool

syntax:`ectool getAllWaitingTasks <projectName> [optionals]`

Example

```
ectool getAllWaitingTasks "proj1" --releaseName "release1"
```

Back to Top

getDeployerApplication

Retrieves the application used in the Release by name.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application. Argument Type: String
`applicationProjectName`	(Optional) The name of the project containing specified application. If this argument is not specified, the default is the Release project. Argument Type: String
`releaseName`	(Optional) Name of the release. Argument type: String

Positional arguments

`projectName`, `applicationName`

Response

Returns a Deployer application object.

ec-perl

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

Example

```
$ec->getDeployerApplication("Default", "Verify versions", {applicationProjectName =
> "Quarterly Summaries"});
```

ectool

syntax: `ectool getDeployerApplication <projectName> <applicationName> [optionals]`

Example

```
ectool getDeployerApplication "Default" "Verify versions" --applicationProjectName
"Quarterly Summaries"
```

Back to Top

getDeployerApplications

Retrieves all the applications in a Release.

You must specify the `projectName` argument.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`releaseName`	(Optional) The name of the Release. Argument Type: String

Positional arguments

`projectName`

Response

Returns all the Deployer application objects.

ec-perl

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

Example

```
$ec->getDeployerApplications ("Default", {releaseName => "Weekly Build"});
```

ectool

syntax: `ectool getDeployerApplications <projectName> [optionals]`

Example

```
ectool getDeployerApplications "Default" --releaseName "Weekly Build"
```

Back to Top

getDeployerConfiguration

Retrieves a Deployer configuration.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application. Argument Type: String
`stageName`	Name of the pipeline stage attached to a Release if the Release is specified. Argument Type: String
`applicationProjectName`	(Optional) The name of the project containing the application. If this argument is not specified, the default is the release project name. Argument Type: String
`releaseName`	(Optional) The name of the Release. Argument Type: String

Positional arguments

`projectName`, `applicationName`, `stageName`

Response

Returns the Deployer configuration object.

ec-perl

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

Example

```
$ec->getDeployerConfiguration("Default", "Shopping Cart", "QA", {applicationProject
Name => "Quarterly Summaries"});
```

ectool

syntax: `ectool getDeployerConfiguration <projectName> <applicationName> <stageName> [optionals]`

Example

```
ectool getDeployerConfiguration "Default" "Shopping Cart" "QA" --applicationProject
Name "Quarterly Summaries"
```

Back to Top

getDeployerConfigurations

Retrieves all the configurations in the Deployer.

You must specify the `projectName` argument.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`applicationName`	(Optional) Name of the application. Argument Type: String
`applicationProjectName`	(Optional) The name of the project containing the application. If this argument is not specified, the default is the release project name. Argument Type: String
`releaseName`	(Optional) The name of the Release. Argument Type: String
`stageName`	(Optional) Name of the pipeline stage attached to a Release if the Release is specified. Argument Type: String

Positional arguments

`projectName`

Response

Returns the list of Deployer configuration objects.

ec-perl

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

Example

```
$ec->getDeployerConfigurations("Default", {applicationProjectName => "Quarterly Sum
maries"});
```

ectool

syntax: `ectool getDeployerConfigurations <projectName> [optionals]`

Example

```
ectool getDeployerConfigurations "Default" --applicationProjectName "Quarterly Summ
aries"
```

Back to Top

getGate

Retrieves a gate by its stage name and gate type.

You must specify the `projectName`, `stageName`, and `gateType`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`stageName`	Name of the stage. Argument Type: String
`gateType`	Type of the gate. Valid values are `PRE` and `POST`. Argument Type: GateType
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String

Positional arguments

`projectName`, `stageName`, `gateType`

Response

Returns a pipeline gate object.

ec-perl

syntax: `$<object>->getGate(<projectName>, <stageName>, <gateType>, {<optionals>});`

Example

```
$ec->getGate("Default", "Preflight", POST, {pipelineName => "Quarterly Summary"});
```

ectool

syntax: `ectool getGate <projectName> <stageName> <gateType> [optionals]`

Example

```
ectool getGate "Default" "Preflight" POST --pipelineName "Quarterly Summary"
```

Back to Top

getNote

Retrieves a note associated with an entity.

You must specify the `noteName` argument.

Arguments	Descriptions
`noteName`	Name of the note. Argument Type: String
`projectName`	(Optional) Project name of the entity that owns the note. Argument Type: String
`releaseName`	(Optional) The name of the Release. Argument Type: String

Positional arguments

`noteName`

Response

Returns the selected note.

ec-perl

syntax: `$<object>->getNote(<noteName>, {<optionals>});`

Example

```
$ec->getNote("Final Approval", {releaseName => "Production"});
```

ectool

syntax: `ectool getNote <noteName> [optionals]`

Example

```
ectool getNote "Final Approval" --releaseName "Production"
```

Back to Top

getNotes

Retrieves all the notes associated with an entity.

Arguments	Descriptions
`projectName`	(Optional) Project name of the entity that owns the note. Argument Type: String

Arguments	Descriptions
`releaseName`	(Optional) The name of the Release. Argument Type: String

Positional arguments

`noteName`

Response

Returns all the notes for a release.

ec-perl

syntax:`$<object>->getNote({<optionals>});`

Example

```
$ec->getNote({releaseName => "Production"});
```

ectool

syntax:`ectool getNote [optionals]`

Example

```
ectool getNote --releaseName "Production"
```

Back to Top

getPipeline

Retrieves a pipeline by its name.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`pipelineName`	Name of the pipeline that must be unique among all projects. Argument Type: String

Positional arguments

`projectName`, `pipelineName`

Response

Returns the specified pipeline element.

ec-perl

syntax: `$<object>->getPipeline(<projectName>, <pipelineName>);`

Example

```
$ec->getPipeline("Default", "Dev to Prod");
```

ectool

syntax: `ectool getPipeline <projectName> <pipelineName>`

Example

```
ectool getPipeline "Default" "Dev to Prod"
```

Back to Top

getPipelineRuntimeDetails

Retrieves pipeline runtime details.

Arguments	Descriptions
`flowRuntimeIds`	(Optional) List of the pipeline flow runtime IDs. Argument Type: Collection

Positional arguments

None

Response

Returns the runtime details of the flow.

ec-perl

syntax: `$<object>->getFlowRuntimeDetails({<optionals>});`

Example

```
$ec->getFlowRuntimeDetails({flowRuntimeIds => "Flow_run_160110" "Flow_run_160112",
"Flow_run_160114"});
```

ectool

syntax: `ectool getFlowRuntimeDetails [optionals]`

Example

```
ectool getFlowRuntimeDetails --flowRuntimeIds "Flow_run_160110" "Flow_run_160112",
"Flow_run_160114"
```

Back to Top

getPipelineRuntimes

Retrieves pipeline runs.

Arguments	Descriptions

`filters`	(Optional) 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 ElectricFlow 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 ElectricFlow 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)` Argument type: Collection
`firstResult`	(Optional) First row of the results to return, based on how the results are paginated. Argument Type: Integer
`flowRuntimeId`	(Optional) ID of the flow runtime. Argument Type: UUID
`maxResults`	(Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer
`pipelineName`	(Optional) Name of the pipeline. Argument Type: String
`projectName`	(Optional) Name of the project that must be unique among all projects. Argument Type: String
`releaseId`	(Optional) ID of the release. Argument Type: String
`sortKey`	(Optional) How to sort the results. Argument Type: String
`sortOrder`	(Optional) The order in which the results are sorted. Argument Type: SortOrder

Positional arguments

None

Response

Returns pipeline run results.

ec-perl

syntax: `$<object>->getPipelineRuntimes({<optionals>});`

Example

```
$ec->getPipelineRuntimes({pipelineName => "Pre-production", projectName => "Defaul
t"});
```

ectool

syntax: `ectool getPipelineRuntimes [optionals]`

Example

```
ectool getPipelineRuntimes --pipelineName "Pre-production" --projectName "Default"
```

Back to Top

getPipelineStageRuntimeDeployerTasks

Retrieves the list of Deployer tasks and their details to be displayed in Pipeline Run Details page.

You must specify the `flowRuntimeId`, `stageName`, and `taskName`.

Arguments	Descriptions
`flowRuntimeId`	ID of the flow runtime. Argument Type: UUID
`stageName`	The name of the stage. Argument Type: String
`taskName`	The name of the Deployer task. Argument Type: String
`firstResult`	(Optional) First row of the results to return, based on how the results are paginated. Argument Type: Integer
`maxResults`	(Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer
`sortKey`	(Optional) How to sort the results. Argument Type: String
`sortOrder`	(Optional) The order in which the results are sorted. Argument Type: SortOrder

Positional arguments

`flowRuntimeId`, `stageName`, `taskName`

Response

Returns a list of Deployer tasks for a pipeline stage and the details about them.

ec-perl

syntax: `$<object>->getPipelineStageRuntimeTasks(<flowRuntimeId>, <stageName>, <taskName>, {<optionals>});`

Example

```
$ec->getPipelineStageRuntimeTasks(4fa765dd-73f1-11e3-b67e-b0a420524165, "PROD", "De
ploy WAR file", {firstResult => 2, maxResults => 200});
```

ectool

syntax: `ectool getPipelineStageRuntimeTasks <flowRuntimeId> <stageName> <taskName> [optionals]`

Example

```
ectool getPipelineStageRuntimeTasks 4fa765dd-73f1-11e3-b67e-b0a420524165 "PROD" "De
ploy WAR file" --firstResult 2 --maxResults 200
```

Back to Top

getPipelineStageRuntimeTasks

Retrieves the list of pipeline stage tasks and the details about them that are displayed in the pipeline run view.

You must specify the `flowRuntimeId`.

Arguments	Descriptions
`flowRuntimeId`	ID of the flow runtime. Argument Type: UUID
`firstResult`	(Optional) First row of the results to return, based on how the results are paginated. Argument Type: Integer
`maxResults`	(Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer
`sortKey`	(Optional) How to sort the results. Argument Type: String
`sortOrder`	(Optional) The order in which the results are sorted. Argument Type: SortOrder
`stageName`	Name of the stage. Argument Type: String

Positional arguments

`flowRuntimeId`

Response

Returns a list of pipeline stage tasks and the details about them in the pipeline run view.

ec-perl

syntax: `$<object>->getPipelineStageRuntimeTasks (<flowRuntimeId>, {<optionals>});`

Example

```
$ec->getPipelineStageRuntimeTasks(4fa765dd-73f1-11e3-b67e-b0a420524165, {firstResul
t => 2, maxResults => 200});
```

ectool

syntax: `ectool getPipelineStageRuntimeTasks <flowRuntimeId> [optionals]`

Example

```
ectool getPipelineStageRuntimeTasks 4fa765dd-73f1-11e3-b67e-b0a420524165 --firstRes
ult 2 --maxResults 200
```

Back to Top

getPipelines

Retrieves all the pipelines.

You must specify the `projectName`.

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

Positional arguments

`projectName`

Response

Returns all of the pipelines in a project.

ec-perl

syntax:`$<object>->getPipelines(<projectName>);`

Example

```
$ec->getPipelines("Default");
```

ectool

syntax: `ectool getPipelines <projectName>`

Example

```
ectool getPipelines "Default"
```

Back to Top

getRelease

Retrieves a Release by name.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`releaseName`	Name of the Release. Argument Type: String

Positional arguments

`projectName`, `releaseName`

Response

Returns the selected Release.

ec-perl

syntax: `$<object>->getRelease(<projectName>, <releaseName>);`

Example

```
$ec->getRelease("Default", "Production");
```

ectool

syntax: `ectool getRelease <projectName> <releaseName>`

Example

```
ectool getRelease "Default" "Production"
```

Back to Top

getReleaseInventory

Retrieves inventory artifacts created in a Release.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`releaseName`	Name of the Release. Argument Type: String

Positional arguments

`projectName`, `releaseName`

Response

Returns a per-Release grouped list of inventory items..

ec-perl

syntax: `$<object>->getReleaseInventory(<projectName>, <releaseName>);`

Example

```
$ec->getReleaseInventory("Default", "Production");
```

ectool

syntax: `ectool getReleaseInventory <projectName> <releaseName>`

Example

```
ectool getReleaseInventory "Default" "Production"
```

Back to Top

getReleases

Retrieves all releases.

You must specify the `projectName` for backward compatibility in ElectricFlow versions before ElectricFlow 6.2.

Arguments	Descriptions
`projectName`	(Optional) Name of the project that must be unique among all projects. Use this as a positional argument in ElectricFlow versions before ElectricFlow 6.2 for backward compatibility. Argument Type: String

Arguments	Descriptions

`filters`	(Optional) 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 ElectricFlow 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 ElectricFlow 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)` Argument type: Collection
`firstResult`	(Optional) First row of the results to return, based on how the results are paginated. Argument Type: Integer
`maxResults`	(Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer
`sortKey`	(Optional) How to sort the results. Argument Type: String
`sortOrder`	(Optional) The order in which the results are sorted. Argument Type: SortOrder

Positional arguments

`projectName` for backward compatibility in ElectricFlow versions before ElectricFlow 6.2.

Response

Returns a list of pipeline stage tasks and the details about them in the pipeline run view.

ec-perl

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

Example

```
$ec->getReleases("Default", {firstResult => 2, maxResults => 200});
```

ectool

syntax: `ectool getReleases <projectName> [optionals]`

Example

```
ectool getReleases "Default" --firstResult 2 --maxResults 200
```

Back to Top

getStage

Retrieves a stage by its name.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`stageName`	Name of the stage. Argument Type: String
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String

Positional arguments

`projectName`, `stageName`

Response

Returns a stage element.

ec-perl

syntax: `$<object>->getStage(<projectName>, <stageName>, {<optionals>});`

Example

```
$ec->getStage("Default", "Preflight", {pipelineName => "Final"});
```

ectool

syntax: `ectool getStage <projectName> <stageName> [optionals]`

Example

```
ectool getStage "Default" "Preflight" --pipelineName "Final"
```

Back to Top

getStages

Retrieves all the stages for a pipeline.

You must specify the `projectName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String

Positional arguments

`projectName`

Response

Returns all the stages for a pipeline.

ec-perl

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

Example

```
$ec->getStages("Default", {pipelineName => "Final"});
```

ectool

syntax: `ectool getStages <projectName> [optionals]`

Example

```
ectool getStages "Default"  --pipelineName "Final"
```

Back to Top

getTask

Retrieves a task by its name.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`taskName`	Name of the task. Argument Type: String
`gateType`	(Optional) Type of the gate. Argument Type: GateType
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String
`stageName`	(Optional) Name of the stage to which this task belongs. Argument type: String

Positional arguments

`projectName`, `taskName`

Response

Returns a task object.

ec-perl

syntax: `$<object>->getTask(<projectName>, <taskName>, {<optionals>});`

Example

```
$ec->getTask("Default", "Check out files", {pipelineName => "Final"});
```

ectool

syntax: `ectool getTask <projectName> <taskName> [optionals]`

Example

```
ectool getTask "Default" "Check out files" --pipelineName "Final"
```

Back to Top

getTasks

Retrieves all tasks.

You must specify the `projectName` argument.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`gateType`	(Optional) Type of the gate. Argument Type: GateType
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String
`stageName`	(Optional) Name of the stage to which this task belongs. Argument type: String

Positional arguments

`projectName`

Response

Returns all the task objects for a pipeline.

ec-perl

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

Example

```
$ec->getTasks("Default", {pipelineName => "Final"});
```

ectool

syntax: `ectool getTasks <projectName> [optionals]`

Example

```
ectool getTasks "Default" --pipelineName "Final"
```

Back to Top

getWaitingTasks

Retrieves a list of all the stage tasks that are currently waiting on manual tasks in pipeline run view.

You must specify the `flowRuntimeId` and `stageName` argument.

Arguments	Descriptions
`flowRuntimeId`	The ID of the flow runtime. Argument type: UUID
`stageName`	Name of the stage. Argument type: String
`deployerTaskName`	(Optional) The name of the deployer task. Argument type: String
`sortKey`	(Optional) How to sort the results: `<jobId|jobName|start|finish|procedureName>`. Argument type: String
`sortOrder`	(Optional) The order in which to sort the results: `<ascending|descending>`. Argument type: SortOrder

Positional arguments

`flowRuntimeId` and `stageName`

Response

Returns all waiting task objects for a pipeline.

ec-perl

syntax: `$<object>->getWaitingTasks(<flowRuntimeId>, <stageName>, {<optionals>});`

Example

```
$ec->getWaitingTasks("5da765dd-73f1-11e3-b67e-b0a420524153", "PROD", {sortKey => st
art, sortOrder => ascending});
```

ectool

syntax: `ectool getWaitingTasks <projectName> <stageName> [optionals]`

Example

```
ectool getWaitingTasks "5da765dd-73f1-11e3-b67e-b0a420524153" "PROD" --sortKey star
t --sortOrder ascending
```

Back to Top

modifyDeployer

Modifies an existing Deployer.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`deployerName`	Name of the Deployer that must be unique within the project. Argument Type: String
`newName`	(Optional) New name for the existing Deployer. Argument Type: String

Positional arguments

`projectName`, `deployerName`

Response

Returns an updated Deployer element.

ec-perl

syntax: `$<object>->modifyDeployer(<projectName>, <deployerName>, {<optionals>});`

Example

```
$ec->modifyDeployer("Default", "Deploy Shopping Cart", {newName => "Shopping Car
t"});
```

ectool

syntax: `ectool modifyDeployer <projectName> <deployerName> [optionals ...]`

Example

```
ectool modifyDeployer "Default" "Deploy Shopping Cart" --newName "Shopping Cart"
```

Back to Top

modifyDeployerApplication

Modifies the Deployer application associated with a Release.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application. Argument Type: String
`applicationProjectName`	(Optional) The name of the project containing specified application. If not specified, it is defaulted to the Release project name. Argument Type: String
`errorHandling`	(Optional) The error handling method for the Deployer application. Argument Type: FlowStateErrorHandling
`orderIndex`	(Optional) The application deployment order, starting at 1. Argument type: Integer
`processName`	(Optional) Name of the process. Argument type: String
`releaseName`	(Optional) Name of the Release. Argument type: String
`smartDeploy`	(Optional) <*Boolean flag -* `0|1|true|false`> If this is set to `1` or `true`, the `smart deploy` option is used when the application is deployed. Argument type: Boolean
`snapshotName`	(Optional) Name of the snapshot. Argument type: String

Arguments	Descriptions
`stageArtifacts`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, artifact staging is enabled for the application process. ElectricFlow retrieves artifacts that will be deployed in an application run before the deployment starts. When this argument is set to `false` or `0`, artifact staging is disabled. Argument Type: Boolean

Positional arguments

`projectName`, `applicationName`

Response

Returns a modified Deployer application and its details.

ec-perl

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

Example

```
$ec->modifyDeployerApplication("Default", "Verify versions", {smartDeploy => 1});
```

ectool

syntax: `ectool modifyDeployerApplication <projectName> <applicationName> [optionals]`

Example

```
ectool modifyDeployerApplication "Default" "Verify versions" --smartDeploy 1
```

Back to Top

modifyDeployerConfiguration

Modifies a Deployer configuration associated with a Deployer application.

You must specify the `projectName`, `applicationName`, and `stageName`.

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

Arguments	Descriptions
`stageName`	Name of the stage of a pipeline that is attached to a Release when a Release is specified. Argument Type: String
`actualParameters`	(Optional) Specifies the list of values to pass as parameters to the flow. 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. Argument Type: Map
`applicationProjectName`	(Optional) The name of the project containing the application. If this argument is not specified, the default is the Release project name. Argument Type: String
`clearActualParameters`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, all actual parameters are removed. Argument type: Boolean
`environmentName`	(Optional) The name of the environment. Argument Type: String
`environmentProjectName`	(Optional) The name of the project containing specified environment or environment template. If this argument is not specified,the default is the Release project name. Argument Type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument Type: String
`environmentTemplateProjectName`	(Optional) Name of the project containing specified environment template. If this argument is not specified, the default is the environment project name. Argument Type: String
`insertRollingDeployManualStep`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` a manual step needs to be added after each phase or batch is run. Argument type: Boolean
`releaseName`	(Optional) The name of the Release. Argument Type: String

Arguments	Descriptions
`rollingDeployEnabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` the pipeline runs the rolling deployment. Argument type: Boolean
`rollingDeployManualStepAssignees`	(Optional) A list of assignees who receive the notification when the rolling deploy iteration is completed. Argument type: Collection
`rollingDeployManualStepCondition`	(Optional) The **Run if** condition (run condition) on the manual step that was created during the rolling deployment. Argument type: NotificationType
`rollingDeployPhases`	(Optional) One or more rolling deploy phases to be used in the rolling deployment. Argument type: Collection
`skipDeploy`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the application is not deployed to an environment. Argument type: Boolean

Positional arguments

`projectName`, `applicationName`, `stageName`

Response

Returns the Deployer configuration and its details.

ec-perl

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

Example

```
$ec->modifyDeployerConfiguration("Default", "Shopping Cart", "Publish components",
{applicationProjectName => "Online store"});
```

ectool

syntax: `ectool modifyDeployerConfiguration <projectName> <applicationName> <stageName> [optionals]`

Example

```
ectool modifyDeployerConfiguration "Default" "Shopping Cart" "Publish components" -
-applicationProjectName "Online store"
```

Back to Top

modifyGate

Modifies an existing gate.

You must specify the `projectName`, `stageName`, and `gateType`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`stageName`	Name of the stage. Argument Type: String
`gateType`	Type of the gate. Valid values are `PRE` and `POST`. Argument Type: GateType
`condition`	(Optional) The **Run if** condition (run condition) is property reference that the pipeline evaluates before executing the next stage. The pipeline waits until one or more dependent run conditions are met. By default, no run conditions are set. The run condition is "fixed text" or text embedding property references that are evaluated into a logical `TRUE` or `FALSE`. An \"0\" or \"false\" is interpreted as `FALSE`. An empty string or any other result is interpreted as `TRUE`. The property reference can be a JavaScript expression, for example, this expression could test whether the name of a step is equal to the value of a property called "restartStep". `$[/javascript (myStep.stepName == myJob.restartStep)]` Argument Type: String
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String

Arguments	Descriptions
`precondition`	(Optional) A property that allows the stage to be created with "pause", which then pauses the pipeline at that point. The pipeline waits until one or more dependent conditions are met. When the pipeline status is eligible to transition from pending to runnable, a **Wait until** condition (precondition)is evaluated. The precondition is fixed text or text embedding a 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 pipeline does not progress until the condition is `TRUE`. By default, this condition is not set and is `FALSE`.. Argument type: String

Positional arguments

`projectName`, `stageName`, `gateType`

Response

Returns a modified gate object.

ec-perl

syntax: `$<object>->modifyGate(<projectName>, <stageName>, <gateType>, {<optionals>});`

Example

```
$ec->modifyGate("Default", "Preflight", POST, {pipelineName => "Annual Summary"});
```

ectool

syntax: `ectool modifyGate <projectName> <stageName> <gateType> [optionals]`

Example

```
ectool modifyGate "Default" "Preflight" POST --pipelineName "Annual Summary"
```

Back to Top

modifyNote

Modifies a note associated with an entity.

You must specify the `noteName` argument.

Arguments	Descriptions
`noteName`	Name of the note. Argument Type: String

Arguments	Descriptions
`newName`	(Optional) New name for the note. Argument Type: String
`note`	(Optional) Notes about the entity. Argument Type: String
`projectName`	(Optional) Project name of the entity that owns the note. Argument Type: String
`releaseName`	(Optional) The name of the Release. Argument Type: String

Positional arguments

`noteName`

Response

Returns a modified note object.

ec-perl

syntax:`$<object>->getNote(<noteName>, {<optionals>});`

Example

```
$ec->getNote("Final Approval", {note => "Not approved until all tests are run"});
```

ectool

syntax:`ectool getNote <noteName> [optionals]`

Example

```
ectool getNote "Final Approval" --note "Not approved until all tests are run"
```

Back to Top

modifyPipeline

Modifies an existing pipeline.

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

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

Arguments	Descriptions
`pipelineName`	Name of the pipeline that must be unique within the project. Argument Type: String
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
`enabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the pipeline is enabled. Argument type: Boolean
`newName`	(Optional) New name for an existing pipeline. Argument Type: String
`type`	(Optional) Type of pipeline. Argument Type: PipelineType

Positional arguments

`projectName`, `pipelineName`

Response

Returns a modified pipeline object.

ec-perl

syntax: `$<object>->modifyPipeline(<projectName>, <pipelineName>, {<optionals>});`

Example

To change the pipeline name:

```
$ec->modifyPipeline("Default", "Web Site Deploy", {newName => "Web Site Update"});
```

ectool

syntax: `ectool modifyPipeline <projectName> <pipelineName> [optionals]`

Example

To change the pipeline name:

```
ectool modifyPipeline "Default" "Web Site Deploy" --newName "Web Site Update"
```

Back to Top

modifyRelease

Modifies an existing Release.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`releaseName`	Name of the release. Argument Type: String
`actualParameters`	(Optional) Specifies the list of values to pass as parameters to the flow. 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. Argument Type: Map
`clearActualParameters`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `true`, tasks remove all actual parameters. Argument type: Boolean
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`. This text is not interpreted by the automation platform. Argument type: String
`newName`	(Optional) New name for an existing pipeline. Argument Type: String
`pipelineName`	(Optional) Name of the pipeline. Argument Type: String
`pipelineProjectName`	(Optional) Name of the project containing specified pipeline. If this argument is not specified, the default is the Release project name. Argument Type: String
`plannedEndDate`	(Optional) The date when this release is expected to end (for example, 2016-05-15). Argument Type: String

Arguments	Descriptions
`plannedStartDate`	(Optional) The date when this release is expected to begin (for example, 2016-05-15) Argument Type: String

Positional arguments

`projectName`, `releaseName`

Response

Returns a modified Release object.

ec-perl

syntax: `$<object>->modifyRelease(<projectName>, <releaseName>, {<optionals>});`

Example

```
$ec->modifyRelease("Default", "Production", {pipelineName => "Q2 Summary"});
```

ectool

syntax: `ectool modifyRelease <projectName> <releaseName> [optionals]`

Example

```
ectool modifyRelease "Default" "Production" --pipelineName "Q2 Summary"
```

Back to Top

modifyStage

Modifies an existing stage.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`stageName`	Name of the stage. Argument Type: String
`afterStage`	(Optional) The stage that is placed after the new stage. Argument Type: String
`beforeStage`	(Optional) The stage that is placed before the new stage. Argument Type: String

Arguments	Descriptions
`condition`	(Optional) The **Run if** condition (run condition) is property reference that the pipeline evaluates before executing to the next stage. The pipeline waits until one or more dependent run conditions are met. By default, no run conditions are set. The run condition is "fixed text" or text embedding property references that are evaluated into a logical `TRUE` or `FALSE`. An \"0\" or \"false\" is interpreted as `FALSE`. An empty string or any other result is interpreted as `TRUE`. The property reference can be a JavaScript expression, for example, this expression could test whether the name of a step is equal to the value of a property called "restartStep". `$[/javascript (myStep.stepName == myJob.restartStep)]` Argument Type: String
`description`	(Optional) Comment text describing this object; it is not interpreted at all by ElectricFlow. Argument Type: String
`newName`	(Optional) New name for the existing stage. Argument Type: String
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String
`precondition`	(Optional) A property that allows the stage to be created with "pause", which then pauses the pipeline at that point. The pipeline waits until one or more dependent conditions are met. When the pipeline status is eligible to transition from pending to runnable, a **Wait until** condition (precondition)is evaluated. The precondition is fixed text or text embedding a 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 pipeline does not progress until the condition is `TRUE`. By default, this condition is not set and is `FALSE`.. Argument type: String

Positional arguments

`projectName`, `stageName`

Response

Returns a modified stage object.

ec-perl

syntax: `$<object>->modifyStage(<projectName>, <stageName>, {<optionals>});`

Example

To modify the name of the stage, enter:

```
$ec->modifyStage("Default", "PROD", {newName => "PROD 2"});
```

To modify the **Wait until** condition (precondition) so that the "Automated Tests" stage is completed before running the "UAT demo" stage, enter:

```
$ec->modifyStage("Default", "UAT demo", {precondition => "$[/myPipelineRuntime/auto
TestCompleted]"});
```

To modify the **Run if** condition (run condition) that the name of the next stage after the "Preflight" stage is "UATdemo", enter:

```
$ec->modifyStage("Default", "Preflight", {condition => "$[/javascript (myPipelineRu
ntime.pipelineName == myPipelineRuntime.UATdemo)]"});
```

ectool

syntax: `ectool modifyStage <projectName> <stageName> [optionals]`

Example

To modify the name of the stage, enter:

```
ectool modifyStage "Default" "PROD" --newName "PROD 2"
```

To modify the **Wait until** condition (precondition) so that the "Automated Tests" stage is completed before running the "UAT demo" stage, enter:

```
ectool modifyStage "Default" "UAT demo" --precondition "$[/myPipelineRuntime/autoTe
stCompleted]"
```

To modify the **Run if** condition (run condition) that the name of the next stage the "Preflights" stage is "UATdemo", enter:

```
ectool modifyStage "Default" "Preflight" --condition => "$[/javascript (myPipelineR
untime.pipelineName == myPipelineRuntime.UATdemo)]"
```

Back to Top

modifyTask

Modifies an existing task.

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

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

Arguments	Descriptions
`taskName`	Name of the task. Argument Type: String
`actualParameters`	(Optional) Specifies the list of values to pass as parameters to the flow. 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. Argument Type: Map
`advancedMode`	(Optional) <*Boolean flag* - `0|1|true|false`> When `advancedMode` is set to `true` or `1`, advanced mode is enabled so that all validations for the `applicationName`, `environmentName`, `processName`, or `snapshotName` will be skipped. These objects can be represented using the "`$[]`" notation and later expanded at the pipeline runtime. For example, when `advancedMode` is enabled, the system does not validate that the environment exists or whether it is mapped to an application. Argument type: Boolean
`afterTask`	(Optional) The task that is placed after the new task. Argument Type: String
`approvers`	(Optional) A list of task approvers who receive the notification. Argument Type: Collection
`beforeTask`	(Optional) The task that is placed before the new task. Argument Type: String
`clearActualParameters`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `true` or `1`, all the actual parameters are removed from the flow. Argument type: Boolean

Arguments	Descriptions
`condition`	(Optional) The **Run if** condition (run condition) is property reference that the pipeline evaluates before executing the next task. The pipeline waits until one or more dependent run conditions are met. By default, no run conditions are set. The run condition is "fixed text" or text embedding property references that are evaluated into a logical `TRUE` or `FALSE`. An \"0\" or \"false\" is interpreted as `FALSE`. An empty string or any other result is interpreted as `TRUE`. The property reference can be a JavaScript expression, for example, this expression could test whether the name of a step is equal to the value of a property called "restartStep". `$[/javascript (myStep.stepName == myJob.restartStep)]`
`description`	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
`enabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the task is enabled. Argument type: Boolean
`environmentName`	(Optional) Name of the environment to create from a template. When the `environmentTemplateName` is specified, the `environmentName` can be specified using the "`$[]`" notation. It is expanded at runtime. Argument Type: String
`environmentProjectName`	(Optional) Name for the project to which the environment or environment template belongs. Argument Type: String
`environmentTemplateName`	(Optional) Name of the environment template. When the `environmentTemplateName` is specified, the `environmentName` can be specified using the "`$[]`" notation. It is expanded at runtime. Argument Type: String
`environmentTemplateProjectName`	(Optional) Name of the project containing specified environment template. If this argument is not specified, the default is the environment project name. Argument Type: String

Arguments	Descriptions
`errorHandling`	(Optional) Type of error handling for this task. Argument Type: FlowStateErrorHandling
`firstTask`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, this task is the first task in the stage, and the `beforeTask` and `afterTask` arguments should not be used. The default is `false` or `0`. Argument type: Boolean
`gateType`	(Optional) The type of the gate. Argument Type: GateType
`insertRollingDeployManualStep`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` a manual step needs to be added after each phase or batch is run. Argument type: Boolean
`instruction`	(Optional) Instruction associated with the task. Argument type: String
`keepOnError`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the system is set to keep environment if an error occurs. The default is `false` or `0`. Argument type: Boolean
`newName`	(Optional) New name for an existing task. Argument Type: String
`notificationTemplate`	(Optional) String containing email formatting instructions for generating notifications. Argument type: String
`pipelineName`	(Optional) Name of the pipeline to which the stage belongs. Argument type: String

Arguments	Descriptions
`precondition`	(Optional) A property that allows the task to be created with "pause", which then pauses the pipeline at that point. The pipeline waits until one or more dependent conditions are met. When the pipeline status is eligible to transition from pending to runnable, a **Wait until** condition (precondition)is evaluated. The precondition is fixed text or text embedding a 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 pipeline does not progress until the condition is `TRUE`. By default, this condition is not set and is `FALSE`.. Argument type: String
`rollingDeployEnabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` the pipeline runs the rolling deployment. Argument type: Boolean
`rollingDeployManualStepAssignees`	(Optional) A list of assignees who receive the notification when the rolling deploy iteration is completed. Argument type: Collection
`rollingDeployManualStepCondition`	(Optional) The **Run if** condition (run condition) on the manual step that was created during the rolling deployment. Argument type: NotificationType
`rollingDeployPhases`	(Optional) One or more rolling deploy phases to be used in the rolling deployment. Argument type: Collection
`skippable`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the task can be skipped in the pipeline. Argument type: Boolean
`snapshotName`	(Optional) Name of the snapshot associated with the application. When `advancedMode` is set to `true` or `1`, advanced mode is enabled so that all validations for the `applicationName`, `environmentName`, `processName`, or `snapshotName` will be skipped. These objects can be represented using the "`$[]`" notation and later expanded at the pipeline runtime. For example, when `advancedMode` is enabled, the system does not validate that the environment exists or whether it is mapped to an application. Argument type: String

Arguments	Descriptions
`stageName`	(Optional) Name of the stage to which the task belongs. Argument Type: String
`startTime`	(Optional) The time to begin invoking this task. The time is formatted `hh:mm` using the 24-hours clock (for example, 17:00). Argument Type: String
`subapplication`	(Optional) Name of the application that owns the subprocess. Argument Type: String
`subpluginKey`	(Optional) If this argument references a procedure, the name of the pluginKey for the procedure
`subprocedure`	(Optional) Name of the subprocedure when a procedure is referenced. Argument Type: String
`subprocess`	(Optional) Name of the ElectricFlow process. Argument Type: String
`subproject`	(Optional) Name of the project in which the procedure runs. Argument Type: String
`subworkflowDefinition`	(Optional) Name of the workflow definition in which the workflow definition is referenced. Argument Type: String
`subworkflowStartingState`	(Optional) Name of the starting state in the specified workflow definition. Argument Type: String
`taskProcessType`	(Optional) The type of the process that the task can invoke. Argument Type: TaskProcessType
`taskType`	(Optional) The type of the task. Argument Type: TaskType
`tierResourceCounts`	(Optional) The resource count for each resource template tier. Argument Type: Map
`workspaceName`	(Optional) The name of the workspace. Argument Type: String

Positional arguments

`projectName`, `taskName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->modifyTask(<projectName>, <taskName>, {<optionals>});`

Example

To rename a task:

```
$ec->modifyTask("Default", "Save results", {newName => "Save output"});
```

To rename a task and make it the first task in a stage:

```
$ec->modifyTask("Default", "Enter user name and password", {firstTask => true, newN
ame => "Enter credentials"});
```

To create a task with **Run if** and **Wait until** conditions:

```
$ec->modifyTask("Default", "Save output", {condition => true, precondition => "Ente
r credentials"});
```

ectool

syntax: `ectool modifyTask <projectName> <taskName> [optionals]`

Example

To rename a task:

```
ectool modifyTask "Default" "Save results" --newName "Save output"
```

To rename a task and make it the first task in a stage:

```
ectool modifyTask "Default" "Enter user name and password" --firstTask true --newNa
me "Enter credentials"
```

To create a task with **Run if** and **Wait until** condition:

Back to Top

removeDeployerApplication

Removes a Deployer application for a Release.

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

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

Arguments	Descriptions
`applicationProjectName`	(Optional) Name of the project containing specified application. If it is not specified, the default is the Release project name. Argument Type: String
`releaseName`	(Optional) Name of the Release. Argument type: String

Positional arguments

`projectName`, `applicationName`

Response

Removes a Deployer application and its details.

ec-perl

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

Example

```
$ec->removeDeployerApplication("Default", "Verify versions", {applicationProjectNam
e => "Software tools"});
```

ectool

syntax: `ectool removeDeployerApplication <projectName> <applicationName> [optionals]`

Example

```
ectool removeDeployerApplication "Default" "Verify versions" --applicationProjectNa
me "Software tools"
```

Back to Top

removeDeployerConfiguration

Removes a Deployer configuration associated with a Deployer application.

You must specify the `projectName`, `applicationName`, and `stageName`.

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

Arguments	Descriptions
`stageName`	Name of the stage of a pipeline attached to a release if specified. Argument Type: String
`applicationProjectName`	(Optional) Name of the project containing the application. If this argument is not specified, the default is the Release project name. Argument Type: String
`releaseName`	(Optional) Name of the Release. Argument Type: String

Positional arguments

`projectName`, `applicationName`, `stageName`

Response

Removes the Deployer configuration and its details.

ec-perl

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

Example

```
$ec->removeDeployerConfiguration("Default", "Shopping Cart", "QA", {applicationProj
ectName => "Online store"});
```

ectool

syntax: `ectool removeDeployerConfiguration <projectName> <applicationName> <stageName> [optionals]`

Example

```
ectool removeDeployerConfiguration "Default" "Shopping Cart" "QA" --applicationProj
ectName "Online store"
```

Back to Top

runPipeline

Runs the specified pipeline.

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

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

Arguments	Descriptions
`pipelineName`	Name of the pipeline that must be unique among all projects. Argument Type: String
`actualParameters`	(Optional) Specifies the list of values to pass as parameters to the flow. 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. Argument Type: Map
`credentials`	(Optional) The credentials to be used in the task. Argument type: Collection
`priority`	(Optional) Priority of jobs launched by the workflow. Argument Type: JobPriority
`scheduleName`	(Optional) Name for the schedule for environment reservations that must be unique among all schedules for the project. Argument Type: String
`stagesToRun`	(Optional) One or more stages to run in a pipeline. Argument Type: Collection
`startingStage`	(Optional) Name of the starting stage. Argument Type: String

Positional arguments

`projectName, pipelineName`

Response

Returns the `flowRuntimeId` object.

ec-perl

syntax: `$<object>->runPipeline(<projectName>, <pipelineName>, {<optionals>});`

Example

```
$ec->runPipeline("Default", "Web Site Update", {startingStage => "Green"});
```

ectool

syntax: `ectool runPipeline <projectName> <pipelineName> [optionals]`

Example

```
ectool runPipeline "Default" "Web Site Update" --startingStage "Green"
```

Back to Top

startRelease

Starts a Release.

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

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`releaseName`	Name of the Release. Argument Type: String
`priority`	(Optional) The priority of jobs launched by the release.. Argument Type: JobPriority
`stagesToRun`	(Optional) One or more stages to run in a pipeline. Argument Type: Collection
`startingStage`	(Optional) The name of the starting stage. Argument type: String

Positional arguments

`projectName`, `releaseName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->startRelease(<projectName>, <releaseName>, {<optionals>});`

Example

```
$ec->startRelease("Default", "Production", {startingStage => "Checkout" });
```

ectool

syntax: `ectool startRelease <projectName> <releaseName> [optionals]`

Example

```
ectool startRelease "Default" "Production" --startingStage "Checkout"
```

Back to Top

validateDeployer

Validates the Deployer configuration.

You must specify the `projectName`, `applicationName`, and `stageName`.

Arguments	Descriptions
`projectName`	Name of the project that must be unique among all projects. Argument Type: String
`applicationName`	Name of the application. Argument Type: String
`stageName`	Name of the stage of a pipeline attached to a release if specified. Argument Type: String
`actualParameters`	(Optional) Specifies the list of values to pass as parameters to the flow. 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. Argument Type: Map
`applicationProjectName`	(Optional) The name of the project containing the application. If this argument is not specified, the default is the release project name. Argument Type: String
`environmentName`	(Optional) Name of the environment to create from a template. Argument Type: String
`environmentProjectName`	(Optional) Name of the project containing specified environment or environment template. If this argument is not specified, the default is the release project name. Argument Type: String
`environmentTemplateName`	(Optional) Name of the environment template to use. Argument Type: String
`environmentTemplateProjectName`	(Optional) Name of the project containing specified environment template. If this argument is not specified, the default is the environment project name. Argument Type: String
`insertRollingDeployManualStep`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` a manual step needs to be added after each phase or batch is run. Argument type: Boolean
`releaseName`	(Optional) Name of the Release. Argument Type: String

Arguments	Descriptions
`rollingDeployEnabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` the pipeline runs the rolling deployment. Argument type: Boolean
`rollingDeployManualStepAssignees`	(Optional) A list of assignees who receive the notification when the rolling deploy iteration is completed. Argument type: Collection
`rollingDeployManualStepCondition`	(Optional) The **Run if** condition (run condition) on the manual step that was created during the rolling deployment. Argument type: NotificationType
`rollingDeployPhases`	(Optional) One or more rolling deploy phases to be used in the rolling deployment. Argument type: Collection
`validationType`	(Optional) The type of validation to perform: • `APPROVER` for approver • `PARAM` for parameter • `ENV` for environment • `ALL` for all validations The default is `ALL`. Argument Type: DeployerValidationType

Positional arguments

`projectName`, `applicationName`, `stageName`

Response

None or a status OK message.

ec-perl

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

Example

```
$ec->validateDeployer("Default", "Make WAR file", "Checkin", {releaseName => "Web S
ite"});
```

ectool

syntax: `ectool validateDeployer <projectName> <applicationName> <stageName> [optionals]`

Example

```
ectool validateDeployer "Default" "Make WAR file" "Checkin" --releaseName "Web Sit
e"
```

Back to Top

waitForFlowRuntime

Waits until the pipeline specified by the flow runtime ID is completed or the timeout expires.

You must specify the `flowRuntimeId`.

This command works only with ec-perl.

Arguments	Descriptions
`flowRuntimeId`	The ID of the flow runtime for the pipeline. Argument type: UUID
`timeout`	(Optional) The number of seconds to wait before giving up on a request. The default is 60 seconds. Argument type: Integer

Positional arguments

`flowRuntimeId`

Response

Returns the `flowRuntimeId`.

ec-perl

syntax: `$<object>->waitForFlowRuntime(<flowRuntimeId>, {<optionals>});`

Example

```
$ec->waitForFlowRuntime("4fa765dd-73f1-11e3-b67e-b0a420524153", {timeout => 30});
```

Back to Top

API Commands - Plugin Management

promotePlugin on page 478

uninstallPlugin on page 478

createPlugin

Creates a plugin from an existing project.

You must specify a `key`, `version`, and `projectName`.

Arguments	Descriptions
`key`	Version independent name for the plugin. Argument type: String
`version`	Version of the plugin. Argument type: String
`projectName`	Name of the project. Argument type: String
`author`	(Optional) Name of the plugin author. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`label`	(Optional) Label that appears in plugin lists. Argument type: String

Positional arguments

`key`, `version`, and `projectName`

Response

Returns a plugin object.

ec-perl

syntax: `$cmdr->createPlugin(<key>, <version>, <projectName>, {<optionals>});`

Example

```
$cmdr->createPlugin("SCM-P4", "2.1.3", "Default", {author => "jdoe"});
```

ectool

syntax: `ectool createPlugin <key> <version> <projectName> [optionals]`

Example

```
ectool createPlugin "SCM-P4" "2.1.3" "Default" --author "jdoe"
```

Back to Top

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. Argument type: String

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. Argument type: String

Positional arguments

`pluginName`

Response

One `plugin` element, which includes the plugin ID, name, time created, label, owner, key, version, and so on.

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
`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, through `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 file system. Argument type: String
`disableProjectTracking`	(Optional) <*Boolean flag -* `0|1|true|false`> If this argument is set to `1` or `true` when importing or exporting a project, even if the original project had Change Tracking enabled, make Change Tracking of the newly imported or exported project be disabled from its creation. If you do not need to track changes to the new project, this avoids the Change Tracking overhead that would otherwise slow down the import operation, and also saves having to subsequently disable change tracking of the re-imported project. Argument type: Boolean
`force`	(Option) <*Boolean flag -* `0|1|true|false`> Specifying `0` or `false` causes an existing plugin with the same key and version to be overwritten with the new plugin contents, otherwise an error is returned. Argument type: Boolean

Arguments	Descriptions
`reducedDetailChangeHistory`	(Optional) <*Boolean flag* - `0|1|true|false`> Use this argument for large projects containing over 20,000 audited objects with Change Tracking enabled. When this argument is set to `true` or `1`, ElectricFlow automatically decreases the amount of Change History indexing information that it saves in a large project, reducing the level of detail for Change Tracking-intensive operations in the Change History. This can make it harder to revert an object to a specific state and to find information in the Change History when you are troubleshooting or debugging an issue. Set this argument to `false` or `0` to suppress to this behavior so that ElectricFlow does not change the amount of indexing information for a large project. This will cause the operation to take longer and put more load on the database, but the Change History will have the full details of the entities owned by objects in the project. Argument type: Boolean

Positional arguments

`url`

Response

One `plugin` element.

ec-perl

syntax: `$cmdr->installPlugin(<url>, {optionals});`

Example

```
$cmdr->installPlugin("./myPlugin.jar", {disableProjectTracking => 1})
```

ectool

syntax: `ectool installPlugin <url> [optionals]`

Example

```
ectool installPlugin "./myPlugin.jar" --disableProjectTracking 1
```

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 ElectricFlow

server cannot. Thus, the Plugin Manager, run in the web server context, provides additional information and functionality.

You must specify the `pluginName`.

Arguments	Descriptions
`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. Argument type: String
`author`	(Optional) The author of the plugin. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`label`	(Optional) The name of the plugin as displayed on the Plugins web page. Argument type: String

Positional arguments

`pluginName`

Response

One `plugin` element.

ec-perl

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

Example

```
$cmdr->modifyPlugin('TheWidget', {description => "new description"});
```

ectool

syntax: `ectool modifyPlugin <pluginName> [optionals]`

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. Argument type: String
`promoted`	(Optional) <*Boolean flag -* `0|1|true|false`> The new value of the promoted flag for the specified plugin. The default is `true`, which means the plugin will be promoted. If you want to demote the plugin, use the value of `"0"` or `false`. Argument type: Boolean

Positional arguments

`pluginName`

Response

One `plugin` element, which includes the plugin ID, name, time created, label, owner, key, version, project name, and so on.

ec-perl

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

Example

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

ectool

syntax: `ectool promotePlugin <pluginName> [optionals]`

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. Argument type: String
`timeout`	(Optional) The maximum amount of time to spend waiting for this operation to complete. Argument type: Long

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> [optionals]`

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
`projectName`	The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure that must be unique within the project. Argument type: String
`credentialName`	(Optional) The name of the credential specified in one of these formats: • **relative**(for example, *"cred1"*)–The credential is assumed to be in the project that contains the requested target object. • **absolute**(for example, *"/projects/BuildProject/credentials/cred1"*)–The credential can be from any specified project, regardless of the project for the target object. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`jobNameTemplate`	(Optional) Template used to determine the default name of jobs launched from a procedure. Argument type: String
`resourceName`	(Optional) The name of the default resource or pool used in steps run by this procedure. Argument type: String

Arguments	Descriptions
`timeLimit`	(Optional) 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 or job becomes runnable (all preconditions are satisfied). Argument type: String
`timeLimitUnits`	(Optional) Time limit units are `hours|minutes|seconds`. Argument type: TimeLimitUnits
`workspaceName`	(Optional) The name of the default workspace used in steps run by this procedure. Argument type: String

Positional arguments

`projectName, procedureName`

Response

Returns a procedure element.

ec-perl

syntax: `$<object>->createProcedure(<projectName>, <procedureName>, {<optionals>});`

Example

```
$ec->createProcedure("Default", "Run Build", {resourceName => "Test Machine 1"});
```

ectool

syntax: `ectool createProcedure <projectName> <procedureName> [optionals]`

Example

```
ectool createProcedure "Default" "Run Build" --resourceName "Test Machine 1"
```

Back to Top

createStep

Creates a new procedure step.

Fundamentally, ElectricFlow 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 ElectricFlow 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
`projectName`	The name of a project that must be unique among all projects. Argument type: String
`procedureName`	The name for a procedure that must be unique within the project. Argument type: String
`stepName`	Name of the step that must be unique within the procedure. Argument type: String
`actualParameters`	(Optional) 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. Argument type: Map
`afterProcedureStep`	(Optional) If specified, the procedure step will be placed after the named procedure step. Argument type: String
`alwaysRun`	(Optional) <*Boolean flag* -`0|1|true|false`>. The default is `false`. If this value is set to `1` or `true`, this step will run even if the job is aborted before the step completes. This is a useful argument for running a "cleanup" step that should run whether the job is successful or not. Argument type: Boolean
`beforeProcedureStep`	(Optional) If specified, the procedure step will be placed before the named procedure step. Argument type: String
`broadcast`	(Optional) <*Boolean flag* -`0|1|true|false`>. The default is `false`. If this value is set to `1` or `true`, the same step is run the same step on several resources at the same time. The step is "broadcast" to all resources listed in the `resourceName` argument. This argument is applicable only to command steps. Argument type: Boolean

Arguments	Descriptions
`command`	(Optional) The command to run. This argument is applicable only to command steps. Argument type: String
`comment`	(Optional) The script to execute the functions of this step, which are passed to the step's shell for execution. Argument type: String
`commandFile`	(Optional) **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`	(Optional) 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. Argument type: String
`credentialName`	(Optional) The credential to use for impersonation on the agent. Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String

Arguments	Descriptions
`errorHandling`	(Optional) 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. Argument type: ErrorHandling
`exclusive`	(Optional) <*Boolean flag* -`0|1|true|false`>. If set to `1` or `true`, this step should acquire and retain this resource exclusively. The default is `false`. **Note:** When you set `exclusive`, `exclusiveMode` is set to `"job"`. Argument type: Boolean
`exclusiveMode`	(Optional) 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. Argument type: ExclusiveMode
`logFileName`	(Optional) A custom log file name produced by running the step. By default, ElectricFlow assigns a unique name for this file. Argument type: String

Arguments	Descriptions
`parallel`	(Optional) The value for `parallel` is a <*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. The default is to `false`. Argument type: Boolean
`postProcessor`	(Optional) The name of a program to run after a step completes. This program looks at the step output to find errors and warnings. ElectricFlow 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). Argument type: String
`precondition`	(Optional) 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 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]` Argument type: String

Arguments	Descriptions
`releaseExclusive`	(Optional) <*Boolean flag -* `0|1|true|false`> This 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`. Argument type: Boolean
`releaseMode`	(Optional) 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. • `releaseToJob` - allows a step to promote a "step exclusive" resource to a Job exclusive resource. Argument type: ReleaseMode
`resourceName`	(Optional) Name for the resource that must be unique among all resources. Argument type: String
`shell`	(Optional) 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. Argument type: String
`subprocedure`	(Optional) 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. Argument type: String
`subproject`	(Optional) 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. Argument type: String

Arguments	Descriptions
`timeLimit`	(Optional) 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. Argument type: String
`timeLimitUnits`	(Optional) Specify `hours|minutes|seconds` for time limit units. Argument type: TimeLimitUnits
`workingDirectory`	(Optional) The ElectricFlow 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, ElectricFlow uses the directory it created for the job in the ElectricFlow workspace as the working directory. **Note:** If running a step on a proxy resource, this directory must exist on the proxy target. Argument type: String
`workspaceName`	(Optional) The name of the workspace where the log files for this step will be stored. Argument type: String

Positional arguments

`projectName, procedureName, stepName`

Response

Returns a step element.

ec-perl

syntax: `$<object>->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

```
$ec->createStep("Test Proj", "Run Build", "Common Cleanup", {subprocedure => "Dela
y",
   actualParameter => {actualParameterName => 'Delay Time', value => '5'}});
```

ectool

syntax: `ectool createStep <projectName> <procedureName> <stepName> [optionals]`

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
`projectName`	The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String

Arguments	Descriptions
`procedureName`	The name of the procedure that must be unique within the project. Argument type: String

Positional arguments

```
projectName, procedureName
```

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteProcedure(<projectName>, <procedureName>);`

Example

```
$ec->deleteProcedure("Default", "Take Snapshot");
```

ectool

syntax: `ectool deleteProcedure <projectName> <procedureName>`

Example

```
ectool deleteProcedure "Default" "Take Snapshot"
```

Back to Top

deleteStep

Deletes a step from a procedure.

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

Arguments	Descriptions
`projectName`	The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure that must be unique within the project. Argument type: String
`stepName`	The name of the step that must be unique within the procedure. Argument type: String

Positional arguments

```
projectName, procedureName, stepName
```

Response

None or a status OK message.

ec-perl

syntax: `$<object>->deleteStep(<projectName>, <procedureName>, <stepName>);`

Example

```
$ec->deleteStep("Default", "Run Build", "Compile");
```

ectool

syntax: `ectool deleteStep <projectName> <procedureName> <stepName>`

Example

```
ectool deleteStep "Default" "Run Build" "Compile"
```

Back to Top

getProcedure

Finds a procedure by its name.

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

Arguments	Descriptions
`projectName`	The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure that must be unique within the project. Argument type: String

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 so on.

ec-perl

syntax: `$<object>->getProcedure(<projectName>, <procedureName>);`

Example

```
$ec->getProcedure("Default", "Run Full Build");
```

ectool

syntax: `ectool getProcedure <projectName> <procedureName>`

Example

```
ectool getProcedure "Default" "Run Full Build"
```

Back to Top

getProcedures

Retrieves all procedures in a project.

You must specify the `projectName`.

Arguments	Descriptions
`projectName`	The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String

Positional arguments

`projectName`

Response

One or more `procedure` elements.

ec-perl

syntax: `$<object>->getProcedures(<projectName>);`

Example

```
$ec->getProcedures("Quarterly Summary");
```

ectool

syntax: `ectool getProcedures <projectName>`

Example

```
ectool getProcedures "Quarterly Summary"
```

Back to Top

getStep

Retrieves a step from a procedure.

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

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure that must be unique within the project. Argument type: String
`stepName`	The name of the step that must be unique within the procedure. Argument type: String

Positional arguments

`projectName, procedureName, stepName`

Response

One `step` element.

ec-perl

syntax: `$<object>->getStep(<projectName>, <procedureName>, <stepName>);`

Example

```
$ec->getStep("Default", "Run Build", "Compile");
```

ectool

syntax: `ectool getStep <projectName> <procedureName> <stepName>`

Example

```
ectool getStep "Default" "Run Build" "Compile"
```

Back to Top

getSteps

Retrieves all steps in a procedure.

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

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`procedureName`	The name of the procedure that must be unique within the project. Argument type: String

Positional arguments

`projectName, procedureName`

Response

Zero or more `step` elements.

ec-perl

syntax: `$<object>->getSteps(<projectName>, <procedureName>);`

Example

```
$ec->getSteps("Default", "Run Build");
```

ectool

syntax: `ectool getSteps <projectName> <procedureName>`

Example

```
ectool getSteps "Default" "Run Build"
```

Back to Top

modifyProcedure

Modifies an existing procedure.

You must specify `projectName` and `procedureName`.

Arguments	Descriptions
`projectName`	The name of the project that contains this procedure. The name must be unique among all projects. You must also enter `procedureName`. Argument type: String
`procedureName`	The name of the procedure that must be unique within the project. You must also enter `projectName`. Argument type: String
`credentialName`	(Optional) Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String

Arguments	Descriptions
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`jobNameTemplate`	(Optional) Job name format for jobs created by running this procedure. Argument type: String
`newName`	(Optional) New name of the procedure. Argument type: String
`resourceName`	(Optional) The name of the default resource where steps belonging to this procedure will run. This name can be a resource pool name. Argument type: String
`timeLimit`	(Optional) 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). Argument type: String
`timeLimitUnits`	(Optional) Time limit units are `hours|minutes|seconds`. Argument type: TimeLimitUnits
`workspaceName`	(Optional) The name of the default workspace where job output is stored. Argument type: String

Positional arguments

`projectName, procedureName`

Response

Returns a modified procedure object.

ec-perl

syntax: `$<object>->modifyProcedure(<projectName>, <procedureName>, {<optionals>});`

Example

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

ectool

syntax: `ectool modifyProcedure <projectName> <procedureName> [optionals]`

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**
`projectName`	The name of the project that must be unique among all projects. You must also enter the `procedureName`. Argument type: String
`procedureName`	The name of the procedure that must be unique within the project. You must also enter `projectName`. Argument type: String
`stepName`	The name of the step that must be unique within the procedure. You must also enter `projectName` and `procedureName`. Argument type: String
`actualParameter`	(Optional) 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. Argument type: Map
`afterProcedureStep`	(Optional) If specified, the procedure step will be placed after the named procedure step.. Argument type: String

Arguments	Descriptions
`alwaysRun`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, this step will run even if the job is aborted before the step completes. This is a useful argument for running a "cleanup" step that should run whether the job is successful or not. Argument type: Boolean
`beforeProcedureStep`	(Optional) The procedure step will be placed before the named procedure step if this option is used. Argument type: String
`broadcast`	(Optional) <*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`. Argument type: Boolean
`clearActualParameters`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `true` or `1`, all the actual parameters will be removed from the step. Argument type: Boolean
`command`	(Optional) The step command. Argument type: String
`commandFile`	(Optional) **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.
`comment`	(Optional) The script to execute the functions of this step, which are passed to the step's shell for execution. Argument type: String
`condition`	(Optional) 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. Argument type: String

Arguments	Descriptions
`credentialName`	(Optional) Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`errorHandling`	(Optional) 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. Argument type: ErrorHandling
`exclusive`	(Optional) <*Boolean flag* -`0|1|true|false`>. If set to `1` or `true`, this indicates this step should acquire and retain this resource exclusively. The default is `0` or `false`. When you set `exclusive`, `exclusiveMode` is set to `job`. Argument type: Boolean

Arguments	Descriptions
`exclusiveMode`	(Optional) 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. Argument type: ExclusiveMode
`logFileName`	(Optional) A custom log file name produced by running the step. By default, ElectricFlow assigns a unique name to this file. Argument type: String
`newName`	(Optional) New name of the step. Argument type: String
`parallel`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, this step and all adjacent steps marked with the flag set will run at the same time. Argument type: String
`postProcessor`	(Optional) This command runs in parallel with the main command for the step. It analyzes the log for the step and collects diagnostic information. Argument type: String

Arguments	Descriptions
`precondition`	(Optional) 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 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]` Argument type: String
`postProcessor`	(Optional) The name of a program to run (script) after a step completes. This program looks at the step output to find errors and warnings. ElectricFlow includes a customizable program called "postp" for this purpose. Argument type: String
`releaseExclusive`	(Optional) <*Boolean flag* - `0|1|true|false`> This declares whether or not this step will release its resource, which is currently held exclusively. **Note:** Setting this flag to `1` or `true` is the same as setting `releaseMode` to `"release"`. Argument type: Boolean

Arguments	Descriptions
`releaseMode`	(Optional) 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. • `releaseToJob` - allows a step to promote a Step exclusive resource to a Job exclusive resource. Argument type: ReleaseMode
`resourceName`	(Optional) The name of the resource used by this step. It must be unique among all resources. Argument type: String
`shell`	(Optional) 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. Argument type: String
`subprocedure`	(Optional) The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the `command` or `commandField`. Argument type: String
`subproject`	(Optional) 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. Argument type: String
`timeLimit`	(Optional) The maximum length of time the step is allowed to run. After the time specified, the step will be aborted. Argument type: String
`timeLimitUnits`	(Optional) Units for step time limit: `<hours|minutes|seconds>` Argument type: TimeLimitUnits

Arguments	Descriptions
`workingDirectory`	(Optional) The ElectricFlow 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, ElectricFlow uses the directory it created for the job in the ElectricFlow workspace. Argument type: String
`workspaceName`	(Optional) The name of the workspace used by this step. Argument type: String

Positional arguments

`projectName, procedureName, stepName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->modifyStep(<projectName>, <procedureName>, <stepName>, {<optionals>});`

Example

```
$ec->modifyStep("Default", "Run Build", "Compile", {commandFile => "tempfile.tx
t"});
```

ectool

syntax: `ectool modifyStep <projectName> <procedureName> <stepName> [optionals]`

Example

```
ectool modifyStep "Default" "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
`projectName`	The name of the project that contains this procedure. The name must be unique among all projects. You must also enter `procedureName`. Argument type: String

Arguments	Descriptions
`procedureName`	The name of the procedure that must be unique within the project. You must also enter `projectName`. Argument type: String
`stepName`	The name of the step that must be unique within the procedure. You must also enter `projectName` and `procedureName`. Argument type: String
`beforeStep`	(Optional) 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. Argument type: String

Positional arguments

`projectName, procedureName, stepName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->moveStep(<projectName>, <procedureName>, <stepName>, {<optionals>});`

Example

```
$ec->moveStep("Default", "Run Build", "Get Sources", {beforeStep => "Compile"});
```

ectool

syntax: `ectool moveStep <projectName> <procedureName> <stepName> [optionals]`

Example

```
ectool moveStep "Default" "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 that 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 containing the application process (`processName`).
When you want to create an application process, specify the name of the application (`applicationName`).
The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to create a component process in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component.
When you want to create a component process in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

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 ElectricFlow.

Argument Type: String

`processType`

Description: Defines the type of action performed by the process. Valid values are `DEPLOY`, `OTHER`, or `UNDEPLOY` for a component 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:

To create a component process in a master component:

```
$ec->createProcess('Default', 'Deploy',  {componentName => 'WAR file'});
```

To create a component process in a specific application:

```
$ec->createProcess('Default', 'Deploy',  {componentName => 'WAR file', component
ApplicationName => 'Shopping Cart'});
```

To create an application process:

```
$ec->createProcess('Default', 'Deploy',  {applicationName => 'Shopping Cart'});
```

ectool

Syntax:

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

Example:

To create a component process in a master component:

```
ectool createProcess 'Default' 'Deploy'  --componentName 'WAR file'
```

To create a component process in a specific application:

```
ectool createProcess 'Default' 'Deploy' --componentName 'WAR file' --componentAp
plicationName 'Shopping Cart'
```

To create an application process:

```
ectool createProcess  'Default' 'Deploy'  --applicationName 'Shopping Cart'
```

Back to Top

deleteProcess

Deletes an application or component process.

Required Arguments

`projectName`

Description: Name of the project that 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 containing the application process (`processName`).
When you want to delete an application process, specify the name of the application (`applicationName`).
The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to delete a component process in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component.
When you want to delete a component process in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

To delete a component process in a master component:

```
$ec->deleteProcess('Default', 'Deploy',  {componentName => 'WAR file'});
```

To create a component process in a specific application:

```
$ec->deleteProcess('Default', 'Deploy',  {componentName => 'WAR file', component
ApplicationName => 'Shopping Cart'});
```

To create an application process:

```
$ec->deleteProcess('Default', 'Deploy',  {applicationName => 'Shopping Cart'});
```

ectool

Syntax:

```
ectool deleteProcess <projectName> <processName> [optionals]
```

Example:

To delete a component process in a master component:

```
ectool deleteProcess 'Default' 'Deploy'  --componentName 'WAR file'
```

To create a component process in a specific application:

```
ectool deleteProcess 'Default' 'Deploy' --componentName 'WAR file' --componentAp
plicationName 'Shopping Cart'
```

To create an application process:

```
ectool deleteProcess  'Default' 'Deploy'  --applicationName 'Shopping Cart'
```

Back to Top

getProcess

Retrieves an application or component process.

Required Arguments

`projectName`

Description: Name of the project that must be unique among all projects.

Argument Type: String

`processName`

Description: Name of the process.

Argument Type: String

Optional Arguments

`applicationEntityRevisionId`

Description: The revision ID of the versioned project.

Argument Type: UUID

`applicationName`

Description: Name of the application containing the application process (`processName`).
When you want to retrieve an application process, specify the name of the application (`applicationName`).
The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to retrieve a component process in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component.
When you want to retrieve a component process in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

Returns the specified process object.

ec-perl

Syntax:

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

Example:

To retrieve a component process in a master component:

```
$ec->getProcess('Default', 'Deploy',  {componentName => 'WAR file'});
```

To retrieve a component process in a specific application:

```
$ec->getProcess('Default', 'Deploy',  {componentName => 'WAR file', componentApp
licationName => 'Shopping Cart'});
```

To retrieve an application process:

```
$ec->getProcess('Default', 'Deploy',  {applicationName => 'Shopping Cart'});
```

ectool

Syntax:

```
ectool getProcess <projectName> <processName> [optionals]
```

Example:

To retrieve a component process in a master component:

```
ectool getProcess 'Default' 'Deploy'  --componentName 'WAR file'
```

To retrieve a component process in a specific application:

```
ectool getProcess 'Default' 'Deploy' --componentName 'WAR file' --componentAppli
cationName 'Shopping Cart'
```

To retrieve an application process:

```
ectool getProcess  'Default' 'Deploy'  --applicationName 'Shopping Cart'
```

Back to Top

getProcesses

Retrieves all processes in an application or component.

Required Arguments

`projectName`

Description: Name of the project that must be unique among all projects.

Argument Type: String

Optional Arguments

`applicationEntityRevisionId`

Description: The revision ID of the versioned project.

Argument Type: UUID

`applicationName`

Description: Name of the application where the processes are located.
When you want to retrieve all the application processes, specify the application (`applicationName`). The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to retrieve all the component processes for a specific component, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component.
When you want to retrieve all the component processes in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

Returns the specified process element.

ec-perl

Syntax:

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

Example:

To retrieve all the component process in a master component:

```
$ec->getProcesses('Default', {componentName => 'WAR file'});
```

To retrieve all the component processes in a specific application:

```
$ec->getProcesses('Default', {componentName => 'WAR file', componentApplicationN
ame => 'Shopping Cart'});
```

To retrieve all the application processes in the specified application:

```
$ec->getProcesses('Default', {applicationName => 'Shopping Cart'});
```

ectool

Syntax:

```
ectool getProcesses <projectName> [optionals]
```

Example:

To retrieve a component process in a master component:

```
ectool getProcesses 'Default' --componentName 'WAR file'
```

To retrieve a component process in a specific application:

```
ectool getProcesses 'Default' --componentName 'WAR file' --componentApplicationN
ame 'Shopping Cart'
```

To retrieve an application process in a specific application:

```
ectool getProcesses  'Default' --applicationName 'Shopping Cart'
```

Back to Top

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 containing the application process (`processName`).
When you want to modify an application process, specify the name of the application (`applicationName`).
The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to modify a component process in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component.
When you want to modify a component process in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

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 ElectricFlow.

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. Valid values are `DEPLOY`, `OTHER`, or `UNDEPLOY` for a component 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

Returns an updated process object.

ec-perl

Syntax:

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

Example:

To modify a component process in a master component:

```
$ec->modifyProcess('Default', 'Deploy',  {componentName => 'WAR file', newName =
> 'Daily Update'});
```

To modify a component process in a specific application:

```
$ec->modifyProcess('Default', 'Deploy',  {componentName => 'WAR file', component
ApplicationName => 'Shopping Cart', newName => 'Daily Update'});
```

To modify an application process:

```
$ec->modifyProcess('Default', 'Deploy',  {applicationName => 'Shopping Cart', ne
wName => 'Daily Update'});
```

ectool

Syntax:

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

Example:

To modify a component process in a master component:

```
ectool modifyProcess 'Default' 'Deploy'  --componentName 'WAR file' --newName 'D
aily Update'
```

To modify a component process in a specific application:

```
ectool modifyProcess 'Default' 'Deploy' --componentName 'WAR file' --componentAp
plicationName 'Shopping Cart' --newName 'Daily Update'
```

To modify an application process:

```
ectool modifyProcess  'Default' 'Deploy'  --applicationName 'Shopping Cart' --ne
wName 'Daily Update'
```

Back to Top

runProcess

Runs the specified process.

Required Arguments

`projectName`

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

Argument Type: String

`applicationName`

Description: Name of the application that owns the process; it must be unique among all applications in the project.

Argument Type: String

`processName`

Description: Name of the application process.

Argument Type: String

Optional Arguments

`actualParameter`

Description: Parameters passed as arguments to the process.

Argument Type: Map

`credentials`

Description: Credentials to use in the job.

Argument Type: Collection

`destinationProject`

Description: Project that will own the job.

Argument Type: String

`environmentName`

Description: Name of the environment.

Argument Type: String

`environmentProjectName`

Description: Name of the project to which the environment or environment template belongs.

Argument Type: String

`environmentTemplateName`

Description: Name of the environment template.

Argument Type: String

`environmentTemplateProjectName`

Description: Name of the project containing specified environment template. If this argument is not specified, the default is the environment project name.

Argument Type: String

`environmentTemplateTierMapName`

Description: Name of the environment template tier map.

Argument Type: String

`keepOnError`

Description: Set this flag to "true" to keep the environment when an error occurs .
The `keepOnError` value is <*Boolean flag* -`0|1|true|false`>.
The default is `false` or `0`.

Argument Type: Boolean

`pollInterval`

Description: This argument requires setting a value in *seconds* to determine how often ectool queries the ElectricFlow 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.
If this option is not specified, `runProcedure` returns immediately.
If it is specified, `runProcedure` waits until the job completes.

Argument Type: Integer

`priority`

Description: Priority of the job.

Argument Type: JobPriority

`rollingDeployEnabled`

Description: <*Boolean flag* -`0|1|true|false`>
Set this flag to "1" or "true" to enable rolling deploy.

Argument Type: Boolean

`rollingDeployManualStepAssignees`

Description: The list of assignees who receive the notification when the rolling deploy iteration is completed.

Argument Type: Collection

`rollingDeployManualStep`

Description: <*Boolean flag* -`0|1|true|false`>
Set this flag to "true" or "1" to wait for human intervention when the rolling deployment phase is completed.

Argument Type: Boolean

`rollingDeployManualStepCondition`

Description: The **Run if** condition (run condition) on the manual step that is created during rolling deployment.

Argument Type: NotificationType

`rollingDeployPhases`

Description: One or more rolling deploy phases to be run in the deployment.

Argument Type: Collection

`scheduleName`

Description: Name for the schedule that must be unique among all schedules for the project.

Argument Type: String

`smartDeploy`

Description: <*Boolean flag* -`0|1|true|false`>
Set this flag to "true" to override the actual parameters.

Argument Type: Boolean

`snapshotName`

Description: Name for the snapshot that must be unique among all snapshots for the project.

Argument Type: String

`stageArtifacts`

Description: <*Boolean flag* -`0|1|true|false`>
Set this flag to "1" or "true" to stage all the artifacts being deployed as part of the application process. The default is "true".

Argument Type: Boolean

`tierMapName`

Description: Name of the tier map used to determine where to run the process.

Argument Type: String

`tierResourceCounts`

Description: Resource count for each resource template tier.

Argument Type: Map

`timeout`

Description: 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.

Argument Type: Integer

`validate`

Description: Validates that the application process, tier map, and environment are well-defined and valid before the running the application process. The value is <*Boolean flag* -`0|1|true|false`>. The default is `true`.

Argument Type: Boolean

Response

Returns new job ID.

If the `--pollInterval` option is provided, ElectricFlow wait until the job completes up to a maximum of `--timeout` seconds (if also provided).

ec-perl

Syntax:

```
$<object>->runProcess(<projectName>, <applicationName>, <processName>,{<optional
s>});
```

Example:

```
$ec->runProcess('Default', 'Deploy', 'Check out', {tierMapName => 'QA test'});
```

ectool

Syntax:

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

Example:

```
ectool runProcess 'Default' 'Deploy' 'Check out' --tierMapName 'QA test'
```

Back to Top

API Commands - Process Dependency

createProcessDependency

Creates a dependency between two process steps.

Required Arguments

`projectName`

Description: Name of the project that 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 containing the specified application process (`processName`).
When you want to create a dependency between two application process steps, specify the name of the application (`applicationName`) containing the process. The application name 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. Possible values are `ALWAYS`, `ERROR`, or `SUCCESS`.

Argument Type: BranchType. Possible values are one of the following:

- `ALWAYS`—Branch is taken regardless of the step outcome
- `ERROR`—Branch is taken if the source step outcome is error
- `SUCCESS`—Branch is taken if the source step outcome is success

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to create a dependency between two component process steps in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component containing the process steps.
When you want to create a dependency between two component process steps in a specific application, specify the component (`componentName`) and application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

Returns a process dependency object.

ec-perl

Syntax:

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

Example:

To create a dependency between two component process steps in a master component:

```
$ec->createProcessDependency('Default', 'Deploy', 'Get WAR file', 'Copy WAR fil
e', {componentName => 'WAR file'});
```

To create a dependency between two component process steps in a specific application:

```
$ec->createProcessDependency'Default', 'Deploy', 'Get WAR file', 'Copy WAR fil
e',  {componentName => 'WAR file', componentApplicationName => 'Shopping Car
t'});
```

To create a dependency between two application process steps:

```
$ec->createProcessDependency('Default', 'Deploy', 'Get WAR file', 'Copy WAR fil
e',  {applicationName => 'Shopping Cart'});
```

ectool

Syntax:

```
ectool createProcessDependency <projectName> <processName> <processStepName> <ta
rgetProcessStepName> [optionals]
```

Example:

To create a dependency between two component process steps in a master component:

```
ectool createProcessDependency 'Default' 'Deploy' 'Get WAR file', 'Copy WAR fil
e' --componentName 'WAR file'
```

To create a dependency between two component process steps in a specific application:

```
ectool createProcessDependency 'Default' 'Deploy' 'Get WAR file', 'Copy WAR fil
e' --componentName 'WAR file' --componentApplicationName 'Shopping Cart'
```

To create a dependency between two application process steps:

```
ectool createProcessDependency  'Default' 'Deploy' 'Get WAR file', 'Copy WAR fil
e' --applicationName 'Shopping Cart'
```

Back to Top

deleteProcessDependency

Deletes a dependency between two process steps.

Required Arguments

`projectName`

Description: Name of the project that 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 containing the specified application process (`processName`).
When you want to delete a dependency between two application process steps, specify the name of the application (`applicationName`) containing the process. The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to delete a dependency between two component process steps in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component containing the process steps.
When you want to delete a dependency between two component process steps in a specific application, specify the component (`componentName`) and application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

To delete a dependency between two component process steps in a master component:

```
$ec->deleteProcessDependency('Default', 'Deploy', 'Get WAR file', 'Copy WAR fil
e', {componentName => 'WAR file'});
```

To delete a dependency between two component process steps in a specific application:

```
$ec->deleteProcessDependency('Default', 'Deploy', 'Get WAR file', 'Copy WAR fil
e',  {componentName => 'WAR file', componentApplicationName => 'Shopping Car
t'});
```

To delete a dependency between two application process steps:

```
$ec->deleteProcessDependency('Default', 'Deploy', 'Get WAR file', 'Copy WAR fil
e',  {applicationName => 'Shopping Cart'});
```

ectool

Syntax:

```
ectool deleteProcessDependency <projectName> <processName> <processStepName> <ta
rgetProcessStepName> [optionals]
```

Example:

To delete a dependency between two component process steps in a master component:

```
ectool deleteProcessDependency 'Default' 'Deploy' 'Get WAR file' 'Copy WAR file'
--componentName 'WAR file'
```

To create a dependency between two component process steps in a specific application:

```
ectool deleteProcessDependency 'Default' 'Deploy' 'Get WAR file' 'Copy WAR file'
--componentName 'WAR file' --componentApplicationName 'Shopping Cart'
```

To create a dependency between two application process steps:

```
ectool deleteProcessDependency  'Default' 'Deploy' 'Get WAR file' 'Copy WAR fil
e' --applicationName 'Shopping Cart'
```

Back to Top

getProcessDependencies

Retrieves all dependencies for a process.

Required Arguments

`projectName`

Description: Name of the project that must be unique among all projects.

Argument Type: String

`processName`

Description: Name of the process.

Argument Type: String

Optional Arguments

`applicationEntityRevisionId`

Description: The revision ID of the versioned project.

Argument Type: UUID

`applicationName`

Description: Name of the application containing the specified application process (`processName`).
When you want to retrieve the dependencies between process steps, specify the name of the application (`applicationName`) containing the process. The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to retrieve the dependencies between component process steps in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component containing the process steps.
When you want to retrieve the dependencies between component process steps in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

Returns zero or more process dependency elements, including `processDependencyId`, `processName`, `source`, `sourceProcessStepName`, `target`, and `targetProcessStepName`.

ec-perl

Syntax:

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

Example:

To retrieve the dependencies between two component process steps in a master component:

```
$ec->getProcessDependencies('Default', 'Deploy', {componentName => 'WAR file'});
```

To retrieve the dependencies between two component process steps in a specific application:

```
$ec->getProcessDependencies('Default', 'Deploy', {componentName => 'WAR file', c
omponentApplicationName => 'Shopping Cart'});
```

To retrieve the dependencies between two application process steps:

```
$ec->getProcessDependencies('Default', 'Deploy', {applicationName => 'Shopping C
art'});
```

ectool

Syntax:

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

Example:

To retrieve the dependencies between two component process steps in a master component:

```
ectool getProcessDependencies 'Default' 'Deploy' --componentName 'WAR file'
```

To retrieve the dependencies between two component process steps in a specific application:

```
ectool getProcessDependencies 'Default' 'Deploy' --componentName 'WAR file' --co
mponentApplicationName 'Shopping Cart'
```

To retrieve the dependencies between two application process steps:

```
ectool getProcessDependencies  'Default' 'Deploy' --applicationName 'Shopping Ca
rt'
```

Back to Top

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 containing the specified application process (`processName`). When you want to modify a dependency between two application process steps, specify the name of the application (`applicationName`) containing the process. The application name 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. Possible values are `ALWAYS`, `ERROR`, or `SUCCESS`

Argument Type: BranchType. Possible values are one of the following:

- `ALWAYS`—Branch is taken regardless of the step outcome
- `ERROR`—Branch is taken if the source step outcome is error
- `SUCCESS`—Branch is taken if the source step outcome is success

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to modify a dependency between two component process steps in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component containing the process steps.
When you want to modify a dependency between two component process steps in a specific application, specify the component (`componentName`) and application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

Returns an updated process dependency object.

ec-perl

Syntax:

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

Example:

To modify a dependency between two component process steps in a master component:

```
$ec->modifyProcessDependency('Default', 'Deploy', 'Get WAR file', 'Copy WAR fil
e', {componentName => 'WAR file', branchConditionName => 'Create new WAR fil
e'});
```

To modify a dependency between two component process steps in a specific application:

```
$ec->modifyProcessDependency('Default', 'Deploy', 'Get WAR file', 'Copy WAR fil
e',  {componentName => 'WAR file', componentApplicationName => 'Shopping Cart',
branchConditionName => 'Create new WAR file'});
```

To modify a dependency between two application process steps:

```
$ec->modifyProcessDependency('Default', 'Deploy', 'Get WAR file', 'Copy WAR fil
e',  {applicationName => 'Shopping Cart', branchConditionName => 'Create new WAR
file'});
```

ectool

Syntax:

```
ectool modifyProcessDependency <projectName> <processName> <processStepName> <ta
rgetProcessStepName> [optionals]
```

Example:

To modify a dependency between two component process steps in a master component:

```
ectool modifyProcessDependency 'Default' 'Deploy' 'Get WAR file', 'Copy WAR fil
e' --componentName 'WAR file' --branchConditionName  'Create new WAR file'
```

To modify a dependency between two component process steps in a specific application:

```
ectool modifyProcessDependency 'Default' 'Deploy' 'Get WAR file', 'Copy WAR fil
e' --componentName 'WAR file' --componentApplicationName 'Shopping Cart' --branc
hConditionName  'Create new WAR file'
```

To modify a dependency between two application process steps:

```
ectool modifyProcessDependency  'Default' 'Deploy' 'Get WAR file', 'Copy WAR fil
e' --applicationName 'Shopping Cart' --branchConditionName  'Create new WAR fil
e'
```

Back to Top

API Commands - Process Step

Note: Several of the following API commands have context type optional arguments. For example, a step command may reference either a procedure or component.

completeManualProcessStep

Completes a manual process step.

Required Arguments

`jobStepId`

Description: This is the unique ElectricFlow-generated identifier (a UUID) for a job step that is assigned automatically when the job step is created. The system also accepts a job step name assigned to the job step by its name template.

Argument Type: UUID

Optional Arguments

`action`

Description: The type of action to be taken on the manual process step, pipeline stage manual task, or stage gate task. Valid values are `completed` and `failed`.

Argument Type: ManualProcessStepStatus

`actualParameters`

Description: The parameters passed as arguments to the process step.

Argument Type: Map

`comment`

Description: Evidence provided while taking an action on the manual process step, which is not interpreted by ElectricFlow.

Argument Type: String

Response

None or status OK message.

ec-perl

Syntax:

```
$<object>->completeManualProcessStep(<jobStepId>, {<optionals>});
```

Example:

To approve the manual process step:

```
$ec->completeManualProcessStep("4fa765dd-73f1-11e3-b67e-b0a420524153", {action =
> completed, comment => "Success"});
```

To reject the manual process step:

```
$ec->completeManualProcessStep("4fa765de-73f1-11e3-b67e-b0a420524153", {action =
> failed, comment => "Try again"});
```

ectool

Syntax:

```
ectool completeManualProcessStep <jobStepId>  [optionals]
```

Example:

To approve the manual process step:

```
ectool completeManualProcessStep "4fa765dd-73f1-11e3-b67e-b0a420524153" --action
completed --comment "Success"
```

To reject the manual process step:

```
ectool completeManualProcessStep "4fa765de-73f1-11e3-b67e-b0a420524153" --action
failed --comment "Try again"
```

Back to Top

createProcessStep

Creates a new process step.

Required Arguments

`projectName`

Description: Name for the project that 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

`actualParameter`

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 containing the specified application process step (`processStepName`).
When you want to create an application process step, specify the name of the application (`applicationName`) containing this process step. The application name must be unique among all projects.

Argument Type: String

`applicationTierName`

Description: Application tier on which to run the step.

Argument Type: String

`assignees`

Description: A list of assignees who receive the notification.

Argument Type: Collection

`beforeProcessStep`

Description: If specified, the process step will be placed before the named process step.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to create a component process step in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component containing the process step.
When you want to create a component process step in a specific application, specify the component (`componentName`) and application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentRollback`

Description: <*Boolean flag* - `0|1|true|false`> When set to `1` or `true` (enabled), rollback will be performed only for the components that were not deployed successfully.

Argument Type: Boolean

`credentialName`

Description: Name of the credential object.

Argument Type: String

`dependencyJoinType`

Description: Join type for incoming dependencies.

Argument Type: ProcessDependencyJoinType

`description`

Description: Comment text describing this object; not interpreted at all by ElectricFlow.

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

`instruction`

Description: Instruction associated with the manual process step.

Argument Type: String

`notificationTemplate`

Description: Email notification template for process step.

Argument Type: String

`processStepType`

Description: Defines type of the process step. For a component process, valid values are command, component, manual, none, plugin, procedure, and utility. For an application process, valid values are command, manual, none, plugin, procedure, process, rollback, and utility.

Argument Type: ProcessStepType

`rollbackSnapshot`

Description: Name of the snapshot to be used for rollback.

Argument Type: String

`rollbackType`

Description: The type of the rollback.Valid values are `environment` (rollback to the previous environment state) and `snapshot` (rollback to a user-specified snapshot).

Argument Type: RollbackProcessStepType

`rollbackUndeployProcess`

Description: Name of the undeploy process to be used during rollback.

Argument Type: String

`smartRollback`

Description: <*Boolean flag* - `0|1|true|false`> The smart deploy flag to be used during rollback is triggered.
When this argument is set to `true` or `1`, smart deploy is enabled during rollback.

Argument Type: Boolean

`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 when it is not 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

Returns a process step element.

ec-perl

Syntax:

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

Example:

To create a component process step in a master component:

```
$ec->createProcessStep('Default', 'Deploy', 'Check out', {componentName => 'WAR
file'});
```

To create a component process step in a specific application:

```
$ec->createProcessStep('Default', 'Deploy', 'Check out', {componentName => 'WAR
file', componentApplicationName => 'Shopping Cart'});
```

To create an application process step:

```
$ec->createProcessStep('Default', 'Deploy', 'Check out', {applicationName => 'Sh
opping Cart'});
```

ectool

Syntax:

```
ectool createProcessStep <projectName> <processName> <processStepName> [optional
s]
```

Example:

To create a component process step in a master component:

```
ectool createProcessStep 'Default' 'Deploy' 'Check out' --componentName 'WAR fil
e'
```

To create a component process step in a specific application:

```
ectool createProcessStep 'Default' 'Deploy' 'Check out' --componentName 'WAR fil
e' --componentApplicationName 'Shopping Cart'
```

To create an application process step:

```
ectool createProcessStep  'Default' 'Deploy' 'Check out' --applicationName 'Shop
ping Cart'
```

Back to Top

deleteProcessStep

Deletes an application or component process step.

Required Arguments

`projectName`

Description: Name for the project that 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 containing the specified application process step (`processStepName`).
When you want to delete an application process step, specify the name of the application (`applicationName`) containing this process step. The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to delete a component process step in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component containing the process step.
When you want to delete a component process step in a specific application, specify the component (`componentName`) and application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

To delete a component process step in a master component:

```
$ec->deleteProcessStep('Default', 'Deploy', 'Check out', {componentName => 'WAR
file'});
```

To delete a component process step in a specific application:

```
$ec->deleteProcessStep('Default', 'Deploy', 'Check out', {componentName => 'WAR
file', componentApplicationName => 'Shopping Cart'});
```

To delete an application process step:

```
$ec->deleteProcessStep('Default', 'Deploy', 'Check out', {applicationName => 'Sh
opping Cart'});
```

ectool

Syntax:

```
ectool deleteProcessStep <projectName> <processName> <processStepName> [optional
s]
```

Example:

To delete a component process step in a master component:

```
ectool deleteProcessStep 'Default' 'Deploy' 'Check out' --componentName 'WAR fil
e'
```

To delete a component process step in a specific application:

```
ectool deleteProcessStep 'Default' 'Deploy' 'Check out' --componentName 'WAR fil
e' --componentApplicationName 'Shopping Cart'
```

To delete an application process step:

```
ectool deleteProcessStep  'Default' 'Deploy' 'Check out' --applicationName 'Shop
ping Cart'
```

Back to Top

getProcessStep

Retrieves an application or component process step.

Required Arguments

`projectName`

Description: Name for the project that 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

`applicationEntityRevisionId`

Description: Revision ID of the versioned object.

Argument Type: UUID

`applicationName`

Description: Name of the application containing the specified application process step (`processStepName`).
When you want to retrieve an application process step, specify the name of the application (`applicationName`) containing this process step. The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to retrieve a component process step in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component containing the process step.
When you want to retrieve a component process step in a specific application, specify the component (`componentName`) and application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

Returns a process step element.

ec-perl

Syntax:

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

Example:

To retrieve a component process step in a master component:

```
$ec->getProcessStep('Default', 'Deploy', 'Check out', {componentName => 'WAR fil
e'});
```

To retrieve a component process step in a specific application:

```
$ec->getProcessStep('Default', 'Deploy', 'Check out', {componentName => 'WAR fil
e', componentApplicationName => 'Shopping Cart'});
```

To retrieve an application process step:

```
$ec->getProcessStep('Default', 'Deploy', 'Check out', {applicationName => 'Shopp
ing Cart'});
```

ectool

Syntax:

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

Example:

To retrieve a component process step in a master component:

```
ectool getProcessStep 'Default' 'Deploy' 'Check out' --componentName 'WAR file'
```

To retrieve a component process step in a specific application:

```
ectool getProcessStep 'Default' 'Deploy' 'Check out' --componentName 'WAR file'
--componentApplicationName 'Shopping Cart'
```

To retrieve an application process step:

```
ectool getProcessStep  'Default' 'Deploy' 'Check out' --applicationName 'Shoppin
g Cart'
```

Back to Top

getProcessSteps

Retrieves all the process steps in an application or component process.

Required Arguments

`projectName`

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

Argument Type: String

`processName`

Description: Name of the process.

Argument Type: String

Optional Arguments

`applicationEntityRevisionId`

Description: Revision ID of the versioned project.

Argument Type: UUID

`applicationName`

Description: Name of the application containing the application process steps.
When you want to retrieve all the application process steps, specify the name of the application (`applicationName`) containing them. The application name must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: Name of the application where the component (`componentName`) is located.
When you want to retrieve all the component process steps in a specific application, specify the component (`componentName`) and application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component containing the process step.
When you want to retrieve all component process steps in a specific application, specify the component (`componentName`) and application (`componentApplicationName`) where the component is located.

Argument Type: String

Response

Returns zero or more process step elements.

ec-perl

Syntax:

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

Example:

To retrieve all the component process steps in a master component:

```
$ec->getProcessSteps('Default', 'Deploy',  {componentName => 'WAR file'});
```

To retrieve all the component process steps in a specific application:

```
$ec->getProcessSteps('Default', 'Deploy',  {componentName => 'WAR file', compone
ntApplicationName => 'Shopping Cart'});
```

To retrieve all the application process steps in an application:

```
$ec->getProcessSteps('Default', 'Deploy',  {applicationName => 'Shopping Car
t'});
```

ectool

Syntax:

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

Example:

To retrieve all the component process steps in a master component:

```
ectool getProcessSteps 'Default' 'Deploy' --componentName 'WAR file'
```

To retrieve all the component process steps in a specific application:

```
ectool getProcessSteps 'Default' 'Deploy'  --componentName 'WAR file' --componen
tApplicationName 'Shopping Cart'
```

To retrieve all the application process steps in an application:

```
ectool getProcessSteps  'Default' 'Deploy' --applicationName 'Shopping Cart'
```

Back to Top

modifyProcessStep

Modifies an existing process step.

Required Arguments

`projectName`

Description: Name of the project that 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

`actualParameter`

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 containing the specified application process step (`processStepName`).
When you want to modify an application process step, specify the name of the application (`applicationName`) containing this process step. The application name 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

`assignees`

Description: A list of assignees who receive the notification.

Argument Type: Collection

`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: Name of the application where the component (`componentName`) is located.
When you want to modify a component process step in a specific application, specify the component (`componentName`) and the application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentName`

Description: Name of the component containing the process step.
When you want to modify a component process step in a specific application, specify the component (`componentName`) and application (`componentApplicationName`) where the component is located.

Argument Type: String

`componentRollback`

Description: <*Boolean flag* - `0|1|true|false`> When set to `1` or `true` (enabled), rollback will be performed only for the components that were not deployed successfully.

Argument Type: Boolean

`credentialName`

Description: Name of the credential object.

Argument Type: String

`dependencyJoinType`

Description: Join type for incoming dependencies.

Argument Type: ProcessDependencyJoinType

`description`

Description: Comment text describing this object; not interpreted at all by ElectricFlow.

Argument Type: String

`errorHandling`

Description: Specifies error handling for this step.

Argument Type: ErrorHandling

`includeCompParameterRef`

Description: <*Boolean flag* - `0|1|true|false`> Set this flag to "true" or "1" if the actual parameters should be generated from component properties. This works only for artifact components.

Argument Type: Boolean

`instruction`

Description: Instruction associated with the manual process step.

Argument Type: String

`newName`

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

Argument Type: String

`notificationTemplate`

Description: Email notification template for process step.

Argument Type: String

`processStepType`

Description: Defines type of the process step. For a component process, valid values are command, component, manual, none, plugin, procedure, and utility. For an application process, valid values are command, manual, none, plugin, procedure, process, rollback, and utility.

Argument Type: ProcessStepType

`rollbackSnapshot`

Description: Name of the snapshot to be used for rollback.

Argument Type: String

`rollbackType`

Description: The type of the rollback.

Argument Type: RollbackProcessStepType

`rollbackUndeployProcess`

Description: Name of the undeploy process to be used during rollback.

Argument Type: String

`smartRollback`

Description: <*Boolean flag* - `0|1|true|false`> The smart deploy flag to be used during rollback is triggered.
When this argument is set to `true` or `1`, smart deploy is enabled during rollback.

Argument Type: Boolean

`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

Returns an updated process step element.

ec-perl

Syntax:

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

Example:

To modify a component process step in a master component:

```
$ec->modifyProcessStep('Default', 'Deploy', 'Check out', {componentName => 'WAR
file', newName => 'Backup files'});
```

To modify a component process step in a specific application:

```
$ec->modifyProcessStep('Default', 'Deploy', 'Check out', {componentName => 'WAR
file', componentApplicationName => 'Shopping Cart', newName => 'Backup files'});
```

To modify an application process step:

```
$ec->modifyProcessStep('Default', 'Deploy', 'Check out', {applicationName => 'Sh
opping Cart', newName => 'Backup files'});
```

ectool

Syntax:

```
ectool modifyProcessStep <projectName> <processName> <processStepName> [optional
s]
```

Example:

To modify a component process step in a master component:

```
ectool modifyProcessStep 'Default' 'Deploy' 'Check out' --componentName 'WAR fil
e' --newName 'Backup files'
```

To modify a component process step in a specific application:

```
ectool modifyProcessStep 'Default' 'Deploy' 'Check out' --componentName 'WAR fil
e' --componentApplicationName 'Shopping Cart' --newName 'Backup files'
```

To modify an application process step:

```
ectool modifyProcessStep  'Default' 'Deploy' 'Check out' --applicationName 'Shop
ping Cart' --newName 'Backup files'
```

Back to Top

API Commands - Project Management

createProject

Creates a new project.

You must specify a `projectName`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`credentialName`	(Optional) Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`resourceName`	(Optional)The name of the resource to use as the default for steps run by procedures in this project. Argument type: String
`tracked`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, Change Tracking is enabled for the project. Argument type: Boolean

Arguments	Descriptions
`workspaceName`	(Optional) The name of a workspace to use as the default for steps run by procedures in this project. Argument type: String

Positional arguments

`projectName`

Response

Returns a project element.

ec-perl

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

Example

```
$cmdr->createProject("Default", {workspaceName => "Primary"});
```

ectool

syntax: `ectool createProject <projectName> [optionals]`

Example

```
ectool createProject "Default" --workspaceName "Primary"
```

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 that must be unique among all projects. Argument type: String
`foreground`	(Optional) <*Boolean flag* - `0|1|true|false`> If the value is `1` or `true`, the object in the foreground is deleted. The default is to delete the object in the background. Argument type: Boolean

Positional arguments

`projectName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteProject("Default", {foreground => true});
```

ectool

syntax: `ectool deleteProject <projectName> [optionals]`

Example

```
ectool deleteProject "Default" --foreground true
```

Back to Top

getProject

Finds a project by its name.

You must specify a `projectName`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String

Positional arguments

`projectName`

Response

One `project` element.

ec-perl

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

Example

```
$cmdr->getProject("Quarterly Summary");
```

ectool

syntax: `ectool getProject <projectName>`

Example

```
ectool getProject "Quarterly Summary"
```

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
`projectName`	The name of the project that must be unique among all projects. Argument type: String

Arguments	Descriptions
`credentialName`	(Optional) Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`newName`	(Optional) New name of the project. Argument type: String
`resourceName`	(Optional) The name of the resource used as the default for steps run by procedures in this project. Argument type: String
`tracked`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, Change Tracking is enabled for the project. Argument type: Boolean
`workspaceName`	(Optional) The name of the default workspace where job output is stored. Argument type: String

Positional arguments

`projectName`

Response

Returns a modified project object.

ec-perl

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

Examples

To change the project description:

```
$cmdr->modifyProject("Default", {description => "A very simple project"});
```

To enable Change Tracking for the Default project:

```
$cmdr->modifyProject("Default", {tracked => true});
```

ectool

syntax: `ectool modifyProject <projectName> [optionals]`

Examples

To change the project description:

```
ectool modifyProject "Default" --description "A very simple project"
```

To enable Change Tracking for the Default project:

```
ectool modifyProject "Default" --tracked true
```

Back to Top

reloadSetupScripts

Runs new, modified, or previously unsuccessful setup scripts.

Arguments	Descriptions
None	–

Positional arguments

None

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool reloadSetupScripts`

Example

```
ectool reloadSetupScripts
```

Back to Top

API Commands - Property Management

deleteProperty on page 550

expandString on page 555

getProperties on page 567

getProperty on page 572

incrementProperty on page 578

modifyProperty on page 584

setProperty on page 590

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.

IMPORTANT: Note: The names "`properties`" and "`project`" are not valid property names.

Arguments	Descriptions
`propertyName`	The name of the property that must be unique within the property sheet. It can be a relative or absolute property path, including "my" paths such as "`/myProject/prop1`". Argument type: String
`applicationName`	(Optional) The name of the application container of the property sheet which owns the property. The name must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier container of the property sheet which owns the property. Argument type: String
`artifactName`	(Optional) The name of the artifact container of the property sheet which owns the property. Argument type: String

Arguments	Descriptions
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String
`componentName`	(Optional) The name of the component container of the property sheet which owns the property. Argument type: String
`configName`	(Optional) The name of the email configuration container that owns the property. Argument type: String
`counter`	(Optional) <*Boolean flag* - `0|1|true|false`> If `1` or `true`, the property is used as a counter. Argument type: Boolean
`credentialName`	(Optional) The name of the credential container of the property sheet which owns the property. Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`credentialProtected`	(Optional) <*Boolean flag* - `0|1|true|false`> If `1` or `true`, permissions required for modify privileges on the property sheet will also require execute privileges on the credentials attached to the property sheet owner. Argument type: Boolean

Arguments	Descriptions
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`environmentName`	(Optional) The name of the environment container of the property sheet that owns the property. The name must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) The name of the environment template container of the property sheet that owns the property. Argument type: String
`environmentTemplateTierName`	(Optional) The name of the environment template tier container of the property sheet that owns the property.
`environmentTierName`	(Optional) The name of the environment tier container of the property sheet that owns the property. Argument type: String
`expandable`	(Optional) Whether or not the property is recursively expandable. <*Boolean flag* - `0|1|true|false`> This determines whether the property value will undergo property expansion when it is fetched. The default is `true`. Argument type: Boolean
`extendedContextSearch`	(Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property. Argument type: Boolean
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String

Arguments	Descriptions
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway container of the property sheet. Argument type: String
`groupName`	(Optional) The name of the group container of the property sheet which owns the property. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notifierName`	(Optional) The name of the email notifier. Argument type: UUID
`objectId`	(Optional) The object identifier returned by `findObjects` and `getObjects`. Argument type: String
`path`	(Optional) Property path string. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	The name of the plugin container of the property sheet which owns the property. Argument type: String

Arguments	Descriptions
`procedureName`	The name of the procedure container of the property sheet which owns the property. When using this argument, you must also enter `projectName`. Argument type: String
`processName`	(Optional) The name of the process when the container is a process or process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the container is a process step. Argument type: String
`projectName`	(Optional) The name of the project container of the property sheet which owns the property. The name must be unique among all projects. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`propertyType`	(Optional) Type of property: <`string|sheet`> This indicates whether to create a string property or a sub-sheet. The default is `string`. Argument type: PropertyType
`releaseName`	(Optional) The name of the Release container of the property sheet which owns the property. Argument type: String
`repositoryName`	(Optional) The name of the repository container of the property sheet which owns the property. Use the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of the resource container of the property sheet which owns the property. You define the new property in this resource. Argument type: String
`resourcePoolName`	(Optional) The name of the resource pool (with one or more resources) container of the property sheet that owns the property. Argument type: String

Arguments	Descriptions
`resourceTemplateName`	(Optional) The name of the resource template container of the property sheet that owns the property. Argument type: String
`scheduleName`	(Optional) The name of the schedule container of the property sheet. If you use a schedule name to define the location for the new property, you must also enter `projectName`. Argument type: String
`snapshotName`	(Optional) The name of the snapshot container of the property sheet which owns the property. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition container of the property sheet which owns the property.
`stateName`	(Optional) The name of the state container of the property sheet which owns the property Argument type: String
`stepName`	(Optional) The name of the step container of the property sheet which owns the property. If you use a step name to define the location for the new property, you must also enter `projectName` and `procedureName`. Argument type: String
`systemObjectName`	(Optional) The name of the special system object. In this context, only `server` is legal. Argument type: SystemObjectName
`taskName`	(Optional) The name of the task in a pipeline stage. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition container of the property sheet which owns the property. Argument type: String
`transitionName`	(Optional) The name of the transition container of the property sheet which owns the property. Argument type: String

Arguments	Descriptions
`userName`	(Optional) The name of the user container of the property sheet which owns the property. Argument type: String
`value`	(Optional) For a string property (see `propertyType`), this is the value of the property. For a sheet property, this argument is invalid. Argument type: String
`valueFile`	(Optional) **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`	(Optional) The name of the workflow definition container of the property sheet which owns the property. Argument type: String
`workflowName`	(Optional) The name of the workflow container of the property sheet which owns the property. Argument type: String
`workspaceName`	(Optional) The name of the workspace container of the property sheet Argument type: String
`zoneName`	(Optional) The name of the zone container of the property sheet. Argument type: String

Positional arguments

`propertyName`

Response

An XML stream that echoes the new property, including its ID, which is assigned by the ElectricFlow 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> [optionals]`

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.

IMPORTANT: Note: The names "`properties`" and "`project`" are not valid property names.

Arguments	Descriptions
`propertyName`	The name of the property that must be unique within the property sheet. Argument type: String
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String

Arguments	Descriptions
`configName`	(Optional) The name of the email configuration. Argument type: String
`credentialName`	(Optional) Whether or not the property is recursively expandable. Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) The name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) The name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`extendedContextSearch`	(Optional) 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, ElectricFlow 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, ElectricFlow looks for the property in the project where the procedure resides. 2. If the object specifier is a job step, ElectricFlow looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. The d efault setting is `true`. Argument type: Boolean
`flowName`	(Optional) The name of the flow. Argument type: String

Arguments	Descriptions
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of a group that contains this property. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notifierName`	(Optional) The name of the email notifier. Argument type: String
`objectId`	(Optional) This is an object identifier returned by `findObjects` and `getObjects`. Argument type: String
`path`	(Optional) Property path. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String

Arguments	Descriptions
pluginName	(Optional) The name of a plugin that may contain a property you want to delete. Argument type: String
procedureName	(Optional) The name of the procedure containing the property you want to delete. **Also requires** projectName Argument type: String
processName	(Optional) The name of the process when the container is a process or process step. Argument type: String
processStepName	(Optional) The name of the process step when the container is a process step. Argument type: String
projectName	(Optional) The name of the project that contains the property you want to delete. Argument type: String
propertySheetId	(Optional) The unique identifier for a property sheet, assigned automatically when the property sheet is created. Argument type: UUID
releaseName	(Optional) The name of the Release. Argument type: String
repositoryName	(Optional) The name of the repository for artifact management. Argument type: String
resourceName	(Optional) The name of the resource that contains the property you want to delete. Argument type: String
resourcePoolName	(Optional) The name of a pool containing one or more resources. Argument type: String
resourceTemplateName	(Optional) The name of the resource template. Argument type: String

Arguments	Descriptions
`scheduleName`	(Optional) The name of the schedule containing the property you want to delete. **Also requires** `projectName`. Argument type: String
`snapshotName`	(Optional) The name of the snapshot. Argument type: String
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step containing the property you want to delete. **Also requires** `projectName` and `procedureName`. Argument type: String
`systemObjectName`	(Optional) The name of a special system object. Only "server" is valid in this context. Argument type: SystemObjectName
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The username that contains this property. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String

Arguments	Descriptions
`workspaceName`	(Optional) The name of the workspace. Argument type: String
`zoneName`	(Optional) The name of the zone. Argument type: String

Positional arguments

`propertyName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteProperty("/projects/Sample project/Changeset ID", {pipelineName => "Q2
Summary"});
```

ectool

syntax: `ectool deleteProperty <propertyName> [optionals]`

Example

```
ectool deleteProperty "/projects/Sample project/Changeset ID" --pipelineName "Q2 Su
mmary"
```

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.

IMPORTANT: Note: The names "`properties`" and "`project`" are not valid property names.

Arguments	Descriptions
`value`	The string value to expand in the given context. Argument type: String

Arguments	Descriptions
applicationName	(Optional) The name of the application that must be unique among all projects. Argument type: String
applicationTierName	(Optional) The name of the application tier. Argument type: String
artifactName	(Optional) The name of the artifact. Argument type: String
artifactVersionName	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String
componentName	(Optional) The name of the component. Argument type: String
configName	(Optional) The name of the email configuration. Argument type: String
credentialName	(Optional) The name of the credential container of the property sheet which owns the property. Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
environmentName	(Optional) The name of the environment that must be unique among all projects. Argument type: String
environmentTemplateName	(Optional) Name of the environment template. Argument type: String

Arguments	Descriptions
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of a group where you might expand a string. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notifierName`	(Optional) The name of the email notifier. Argument type: String
`objectId`	(Optional) This is an object identifier returned by `findObjects` and `getObjects`. Argument type: String

Arguments	Descriptions
`path`	(Optional) The property path. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of a plugin where you might expand a string. Argument type: String
`procedureName`	(Optional) The name of a procedure where you might need to expand a string. **Also requires** `projectName`. Argument type: String
`processName`	(Optional) The name of the process when the container is a process or process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the container is a process step. Argument type: String
`projectName`	(Optional) The name of the project that contains the string to expand. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that assigned automatically when the property sheet is created. Argument type: UUID
`releaseName`	(Optional) The name of the Release. Argument type: String
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of a resource where you might expand a string. Argument type: String
`resourcePoolName`	(Optional) The name of a pool containing one or more resource Argument type: String

Arguments	Descriptions
`resourceTemplateName`	(Optional) Name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of a schedule within this project. **Also requires** `projectName`. Argument type: String
`snapshotName`	(Optional) The name of the snapshot. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step whose string you might be expanding. **Also requires** `projectName` and `procedureName`. Argument type: String
`systemObjectName`	(Optional) System object names include: `admin|directory|log|priority|projects|resources|server|session|workspaces`. Argument type: SystemObjectName
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The name of the user where you may need to expand the string. Argument type: String

Arguments	Descriptions
`valueFile`	(Optional) **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`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String
`workspaceName`	(Optional) The name of a workspace where you may need to expand the string. Argument type: String
`zoneName`	(Optional) The name of the zone. Argument type: String

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> [optionals]`

Examples

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

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

Back to Top

findProperties

Returns the properties of the specified object.

You must specify object locators for the properties you want to find.

IMPORTANT: Note: The names "`properties`" and "`project`" are not valid property names.

Arguments	**Descriptions**
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String

Arguments	Descriptions
`credentialName`	(Optional) The name of the credential containing the properties to retrieve. Name of the credential in one of these 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. Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`expand`	(Optional) <*Boolean flag* - `0|1|true|false`> The default value is `1` (`true`), and the value of each property will be expanded. A value of `0` (`false`) will cause the unexpanded value of each property to be returned. Argument type: Boolean
`filters`	(Optional)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. Argument type: Collection
`flowName`	(Optional) The name of the flow. Argument type: String

Arguments	Descriptions
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of the group containing the properties to retrieve. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`maxIds`	(Optional) `<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. Argument type: Integer
`notifierName`	(Optional) The name of the email notifier. Argument type: String

Arguments	Descriptions
`numObjects`	(Optional) `<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. Argument type: Integer
`objectId`	(Optional) This is an object identifier returned by `findObjects` and `getObjects`. Argument type: String
`path`	(Optional) 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. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin containing the properties to retrieve. Argument type: String
`procedureName`	The name of the procedure containing the properties to retrieve. **Also requires** `projectName`. Argument type: String
`processName`	(Optional) The name of the process when the container is a process or process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the container is a process step. Argument type: String
`projectName`	(Optional) The name of the project containing the properties to retrieve. Argument type: String
`releaseName`	(Optional) The name of the Release. Argument type: String

Arguments	Descriptions
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of the resource containing the properties to retrieve. Argument type: String
`resourcePoolName`	(Optional) The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of the schedule containing the properties to retrieve. **Also requires** `projectName`. Argument type: String
`snapshotName`	(Optional) The name of the snapshot. Argument type: String
`sorts`	(Optional) This is an ordered list of sort criteria. This option works only when you specify a property name. 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 ElectricFlow Perl API. Argument type: Collection
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String

Arguments	Descriptions
`stepName`	(Optional) The name of the step containing the properties to retrieve. **Also requires** `projectName` and `procedureName`. Argument type: String
`systemObjectName`	(Optional) The name of the system object containing the properties to retrieve. Only "server" is supported. Argument type: SystemObjectName
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The name of the user containing the properties to retrieve. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition.
`workflowName`	(Optional) The name of the workflow. **Also requires**`projectName`. Argument type: String
`workspaceName`	(Optional) The name of the workspace containing the properties to retrieve. Argument type: String
`zoneName`	(Optional) The name of the zone. Argument type: String

Positional arguments

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

Response

One or more `property` elements for the object.

ec-perl

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

Examples

```
$cmdr->findProperties({resourceName => "Web Server"});
```

ectool

syntax: `ectool findProperties [optionals]`

Examples

```
ectool findProperties --resourceName "Web Server"
```

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.

IMPORTANT: Note: The names "`properties`" and "`project`" are not valid property names.

Arguments	Descriptions
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String

Arguments	Descriptions
`credentialName`	(Optional) The name of the credential containing the properties to retrieve. Name of the credential in one of these 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. Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`expand`	(Optional) <*Boolean flag* - `0|1|true|false`> The default value is `1` (`true`), and the value of each property will be expanded. A value of `0` (`false`) will cause the unexpanded value of each property to be returned. Argument type: Boolean
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String

Arguments	Descriptions
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of the group containing the properties to retrieve. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notifierName`	(Optional) The name of the email notifier. Argument type: String
`objectId`	(Optional) This is an object identifier returned by `findObjects` and `getObjects`. Argument type: String
`path`	(Optional) 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. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin containing the properties to retrieve. Argument type: String

Arguments	Descriptions
`procedureName`	The name of the procedure containing the properties to retrieve. **Also requires** `projectName`. Argument type: String
`processName`	(Optional) The name of the process when the container is a process or process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the container is a process step. Argument type: String
`projectName`	(Optional) The name of the project containing the properties to retrieve. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`recurse`	(Optional) <*Boolean flag* - `0|1|true|false`> The default value is `0` (`false`), and 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`. Argument type: Boolean
`releaseName`	(Optional) The name of the release container of the property sheet which owns the property. Argument type: String
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of the resource containing the properties to retrieve. Argument type: String
`resourcePoolName`	(Optional) The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument type: String

Arguments	Descriptions
`scheduleName`	(Optional) The name of the schedule containing the properties to retrieve. **Also requires** `projectName`. Argument type: String
`snapshotName`	(Optional) The name of the snapshot. Argument type: String
`stageName`	(Optional) The name of the stage. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step containing the properties to retrieve. **Also requires** `projectName` and `procedureName`. Argument type: String
`systemObjectName`	(Optional) The name of the system object containing the properties to retrieve. Only "server" is supported. Argument type: SystemObjectName
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The name of the user containing the properties to retrieve. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition.

Arguments	Descriptions
`workflowName`	(Optional) The name of the workflow. **Also requires** `projectName`. Argument type: String
`workspaceName`	(Optional) The name of the workspace containing the properties to retrieve. Argument type: String
`zoneName`	(Optional) The name of the zone. Argument type: String

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 [optionals]`

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. The name must be unique within the property sheet. **IMPORTANT: Note:** The names "`properties`" and "`project`" are not valid property names. Argument type: String
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String
`credentialName`	(Optional) The name of the credential containing the property to retrieve. Name of the credential in one of these 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. Argument type: String

Arguments	Descriptions
`environmentName`	(Optional) The name of the environment container that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`expand`	(Optional) <*Boolean flag* - `0|1|true|false`> The default value is `1` (`true`), and the value of each property will be expanded. A value of `0` (`false`) will cause the unexpanded value of each property to be returned. Argument type: Boolean
`extendedContextSearch`	(Optional) 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, ElectricFlow 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, ElectricFlow looks for the property in the project where the procedure resides. • If the object locator is a job step, ElectricFlow looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. Default setting is `true`. Argument type: Boolean
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String

Arguments	Descriptions
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of the group containing the property to retrieve. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String
`notifierName`	(Optional) The name of the email notifier. Argument type: String
`objectId`	(Optional) This is an object identifier returned by `findObjects` and `getObjects`. Argument type: String
`path`	(Optional) 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. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin containing the property to retrieve. Argument type: String

Arguments	Descriptions
`procedureName`	(Optional) The name of the procedure containing the property to retrieve. **Also requires** `projectName`. Argument type: String
`processName`	(Optional) The name of the process when the container is a process or process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the container is a process step. Argument type: String
`projectName`	(Optional) The name of the project containing the property to retrieve. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`releaseName`	(Optional) The name of the Release container of the property sheet which owns the property. Argument type: String
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of the resource containing the property to retrieve. Argument type: String
`resourcePoolName`	(Optional) The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of the schedule containing the property to retrieve. **Also requires** `projectName`. Argument type: String

Arguments	Descriptions
`snapshotName`	(Optional) The name of the snapshot. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition.
`stateName`	(Optional) The name of the state.
`stepName`	(Optional) The name of the step containing the property to retrieve. **Also requires** `projectName` and `procedureName`
`systemObjectName`	(Optional) The name of the system object containing the property to retrieve. Only "server" is supported. Argument type: SystemObjectName
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The name of the user containing the property to retrieve. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String
`workspaceName`	(Optional) The name of the workspace containing the property to retrieve. Argument type: String
`zoneName`	(Optional) The name of the zone. Argument type: String

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> [optionals]`

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

Automically 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`.

IMPORTANT: Note: The names "`properties`" and "`project`" are not valid property names.

Arguments	Descriptions
`propertyName`	Name for the property that must be unique within the property sheet. Argument type: String
`incrementBy`	The amount by which to increment the property, which can be a positive or negative integer. Argument type: Integer

Arguments	Descriptions
`applicationName`	(Optional) The name of the application. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String
`credentialName`	(Optional) Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String

Arguments	Descriptions
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`extendedContextSearch`	(Optional) 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 this argument is set to `1` or `true`, and there is an object specified in the command, ElectricFlow first looks for the property in that object specifier, but also searches in other locations if not found, according to the following rules: • If the object specifier is a procedure, ElectricFlow looks for the property in the project where the procedure resides. • If the object specifier is a job step, ElectricFlow looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. The default setting is `true`. Argument type: Boolean
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of the group containing the property to increment. Argument type: String

Arguments	Descriptions
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notifierName`	(Optional) The name of the email notifier. Argument type: String
`objectId`	(Optional) This is an object identifier returned by `findObjects` and `getObjects`. Argument type: String
`path`	(Optional) Path to the property. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin containing a property to increment.
`procedureName`	(Optional) The name of the procedure containing this property. **Also requires** `projectName`. Argument type: String
`processName`	(Optional) The name of the process when the container is a process or process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the container is a process step. Argument type: String
`projectName`	The name of the project containing this property. **Also requires** `procedureName`. Argument type: String

Arguments	Descriptions
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`releaseName`	(Optional) The name of the Release containing this property. Argument type: String
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of the resource containing this property. Argument type: String
`resourcePoolName`	The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	Name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of the schedule containing this property. **Also requires** `projectName`. Argument type: String
`snapshotName`	(Optional) The name of the snapshot. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step containing this property. **Also requires** `projectName` and `procedureName`. Argument type: String
`systemObjectName`	(Optional) Only `server` is a valid system object for this API. Argument type: SystemObjectName

Arguments	Descriptions
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The name of the user containing this property. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String
`workspaceName`	(Optional) The name of the workspace containing this property. Argument type: String
`zoneName`	(Optional) The name of the zone. Argument type: String

Positional arguments

`propertyName, incrementBy`

Response

A text string containing the updated numeric property value.

ec-perl

syntax: `$cmdr->incrementProperty(<propertyName> , <incrementBy>, {<optionals>});`

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> [optionals]`

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`.

IMPORTANT: Note: The names "`properties`" and "`project`" are not valid property names.

Arguments	Descriptions
`propertyName`	The name of the property that must be unique within the property sheet. This argument can be a path. Argument type: String
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String

Arguments	Descriptions
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String
`counter`	(Optional) <*Boolean flag* -`0|1|true|false`> If set to `true` or `1`, this property is used as a counter. Argument type: Boolean
`credentialName`	(Optional) `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. Argument type: String
`credentialProtected`	(Optional) <*Boolean flag* - `0|1|true|false`> If `1` or `true`, permissions required for modify privileges on the property sheet will also require execute privileges on the credentials attached to the property sheet owner. Argument type: Boolean

`description`	(Optional) A plain text or HTML description for this object.
	If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are:
	`<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span>`
	`<style> <table> <tc> <td> <th> <tr> <ul>`
	
	Argument type: String

`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String

Arguments	Descriptions
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`expandable`	(Optional) <*Boolean flag* -`0|1|true|false`>–Determines whether the property will undergo property expansion when it is retrieved. The default is `true`. Argument type: Boolean
`extendedContextSearch`	(Optional) 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, ElectricFlow 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, ElectricFlow looks for the property in the project where the procedure resides. 2. If the object specifier is a job step, ElectricFlow looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. The default setting is `true`. Argument type: Boolean
`gatewayName`	(Optional) Name of the gateway. Argument type: String
`groupName`	(Optional) Name of the group. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step, which is assigned automatically when the job step is created. Argument type: UUID
`newName`	(Optional) New name of the property. Argument type: String
`notifierName`	(Optional) Name of the email notifier. Argument type: String

Arguments	Descriptions
`objectId`	(Optional) An object identifier returned by `findObjects` and `getObjects`. Argument type: String
`path`	(Optional) Path to the property. Argument type: String
`pipelineName`	(Optional) Name of the pipeline. Argument type: String
`pluginName`	(Optional) Name of the plugin. Argument type: String
`procedureName`	(Optional) Name of the procedure. **Also requires** `projectName`. Argument type: String
`processName`	(Optional) Name of the process if the container is a process or process step. Argument type: String
`processStepName`	(Optional) Name of the process step if the container is a process step. Argument type: String
`projectName`	(Optional) Name of the project. The property may be on the project itself or on the object, which is indicated by other arguments. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet, which is assigned automatically when the property sheet is created. Argument type: UUID
`propertyType`	(Optional) `<string|sheet>`– Indicates whether to create a string property or a sub-sheet. The default is `string`. Argument type: PropertyType
`releaseName`	(Optional) Name of the Release. Argument type: String
`repositoryName`	(Optional) Name of the repository for artifact management. Argument type: String

Arguments	Descriptions
`resourceName`	(Optional) Name of the resource. Argument type: String
`resourcePoolName`	(Optional) Name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument type: String
`scheduleName`	(Optional) Name of the schedule. **Also requires** `projectName`. Argument type: String
`snapshotName`	(Optional) The name of the snapshot. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) Name of the state definition. Argument type: String
`stateName`	(Optional) Name of the state. Argument type: String
`stepName`	Name of the step containing the property to be modified. **Also requires** `projectName` and `procedureName`. Argument type: String
`systemObjectName`	System objects include: `admin|artifactVersions|directory|emailConfigs|log|plugins|server|session|workspaces`. Argument type: SystemObjectName
`taskName`	(Optional) Name name of the task. Argument type: String
`transitionDefinitionName`	(Optional) Name of the transition definition. Argument type: String
`transitionName`	(Optional) Name of the transition. Argument type: String

Arguments	Descriptions
`userName`	(Optional) Name of the user.(Optional) Argument type: String
`value`	(Optional) Value of the property. Argument type: String
`valueFile`	(Optional) **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`	(Optional) Name of the workflow definition. Argument type: String
`workflowName`	(Optional) Name of the workflow. Argument type: String
`workspaceName`	(Optional) Name of the workspace. Argument type: String
`zoneName`	(Optional) Name of the zone. Argument type: String

Positional arguments

`propertyName`

Response

An XML stream that echoes the modified property.

ec-perl

syntax: `$cmdr->modifyProperty(<propertyName>, {<optionals>});`

Example

```
$cmdr->modifyProperty("Saved Variables", {description =>
     "Starting configuration of name/value pairs", jobId => 4fa765dd-73f1-11e3-b67
e-b0a420524153});
```

ectool

syntax: `ectool modifyProperty <propertyName> [optionals]`

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 the `propertyName` and `value` arguments. 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.

IMPORTANT: Note: The names "`properties`" and "`project`" are not valid property names.

Arguments	Descriptions
`propertyName`	The name or path of the property that must be unique within the property sheet. This argument can be a path. Argument type: String
`value`	The value of the property. Argument type: 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.
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String

Arguments	Descriptions
`configName`	(Optional) The name of the email configuration. Argument type: String
`counter`	(Optional) <*Boolean flag* - `0|1|true|false`> If the argument is set to `1` or `true`, the property is used as a counter. The default is `0` (`false`). ant the property to expand, use the value of `0` (`false`). Argument type: Boolean
`credentialName`	(Optional) The name of the credential containing the property you want to set. Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. **Also requires** `projectName`. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String

Arguments	Descriptions
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`expandable`	(Optional) <*Boolean flag* - `0|1|true|false`> The default is `1` (`true`),and the property value will be expanded when referenced. If you do not want the property to expand, use the value of `0` (`false`). Argument type: Boolean
`extendedContextSearch`	(Optional) <*Boolean flag* - `0|1|true|false`> If set, and there is an object specified in the command, ElectricFlow first looks for the property in the object specified, but also searches in other locations if not found, according to the following rules: If the object specified is a procedure, ElectricFlow looks for the property in the project where the procedure resides. If the object specified is a job step, ElectricFlow looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties. The default setting is `false`. Argument type: Boolean
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional)The name of the gateway containing the property that you want to set. Argument type: String

Arguments	Descriptions
`groupName`	(Optional) The name of the group containing the property you want to set. Argument type: String
`jobId`	(Optional) The name of the job containing the property you want to set. The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional)The name of the job step containing the property that you want to set. The unique identifier for a job step, assigned automatically when the job step is created. Argument type: UUID
`notifierName`	(Optional) The name of the email notifier. Argument type: String
`objectId`	(Optional) This is an object identifier returned by `findObjects` and `getObjects`. Argument type: String
`path`	(Optional) The path to the property. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin containing the property you want to set. Argument type: String
`procedureName`	(Optional) The name of the procedure containing the property that you want to set. **Also requires** `projectName`.(Optional) Argument type: String
`processName`	(Optional) The name of the process when the container is a process or process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the container is a process step. Argument type: String

Arguments	Descriptions
`projectName`	(Optional) The name of the project containing the property that you want to set. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: String
`releaseName`	(Optional) The name of the Release. Argument type: String
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of the resource containing the property that you want to set. Argument type: String
`resourcePoolName`	(Optional) The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	(Optional) Name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of the schedule containing the property you want to set. **Also requires** `projectName`. Argument type: String
`snapshotName`	(Optional) The name of the snapshot. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String

Arguments	Descriptions
`stepName`	(Optional) The name of the step containing the property you want to set. **Also requires** `projectName` and `procedureName`. Argument type: String
`systemObjectName`	(Optional) 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` Argument type: SystemObjectName
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`transitionName`	(Optional) The name of the transition. Argument type: String
`userName`	(Optional) The name of the user containing the property you want to set. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workflowName`	(Optional) The name of the workflow. Argument type: String
`workspaceName`	(Optional) The name of the workspace containing the property that you want to set. Argument type: String
`zoneName`	(Optional) The name of the zone. Argument type: String

Positional arguments

`propertyName`

Response

An XML stream that echoes the property.

ec-perl

syntax: `$cmdr->setProperty(<propertyName>, {<optionals>});`

Examples

```
$cmdr->setProperty("Changeset ID", {value => "14992", projectName => "Sample Projec
t"});

$cmdr->setProperty("/myResource/Application Path", "c:\Program Files\Application");

$cmdr->setProperty("Application Path", "c:\Program Files\Application",
   {resourceName => "r2"});
```

ectool

syntax: `ectool setProperty <propertyName> [optionals]`

Examples

```
ectool setProperty "Changeset ID" --value "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 the specified resource pool (a named group of resources).

You must specify a `resourcePoolName`.

Arguments	Descriptions
`resourcePoolName`	Name for the resource pool that must be unique among all resource pools. The resource pool has one or more resources. Argument Type: String
`resourceNames`	(Optional) List of resources to add to the resource pool. Argument Type: Collection

Positional arguments

`resourcePoolName`

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->addResourcesToPool("Test Station 1", {resourceNames => ["Server1", "Server
2", "Server3"]});
```

ectool

syntax: `ectool addResourcesToPool <resourcePoolName> [optionals]`

Example

```
ectool addResourcesToPool "Test Station 1" --resourceNames Server1 Server2 Server3
```

Back to Top

addResourceToEnvironmentTier

Adds a resource to the specified environment tier.

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

Arguments	Descriptions
`resourceName`	Name for the resource that must be unique among all resources. Argument Type: String
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`environmentName`	Name of the environment that must be unique among all projects. Argument Type: String
`environmentTierName`	Name for the environment tier that must be unique among all tiers for the environment. Argument Type: String
`rollingDeployPhaseName`	(Optional) Name for the rolling deploy phase to be associated with the resource. Argument Type: String

Positional arguments

`resourceName, projectName, environmentName, environmentTierName`

Response

None or a status OK message.

ec-perl

syntax:`$<object>->addResourceToEnvironmentTier(<resourceName>, <projectName>, <environmentName>, <environmentTierName>, {<optionals>});`

Example

```
$ec->addResourceToEnvironmentTier("Web Server", "Default", "DEV", "DB Server", {
rollingDeployPhaseName => "Blue UAT demo"});
```

ectool

syntax:`addResourceToEnvironmentTier <resourceName> <projectName> <environmentName> <environmentTierName> [optionals]`

Example

```
ectool addResourceToEnvironmentTier "Web Server" "Default" "DEV" "DB Server" --r
ollingDeployPhaseName "Blue UAT demo"
```

Back to Top

createResource

Creates a new resource.

IMPORTANT: For a proxy resource, the `proxyHostName` and `proxyPort` arguments refer to the proxying ElectricFlow agent. The `hostName` and `port` arguments refer to the proxy target.

You must specify a `resourceName`.

Arguments	Descriptions
`resourceName`	Name of the new resource that must be unique among all resources. Argument Type: String
`artifactCacheDirectory`	(Optional) Directory on the agent host where retrieved artifacts are stored. Argument Type: String
`block`	(Optional) <*Boolean flag* - `0|1|true|false`> A newly created resource will be pinged. This argument makes the `createResource` call block until the result of the ping is known. The default is *false*. Argument Type: Boolean
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument Type: String
`hostName`	(Optional) If it is an ordinary resource, the name or IP address of the machine containing the ElectricFlow agent for this resource . If this is a proxy resource, the name or IP address of the proxy target. Argument Type: String
`hostType`	(Optional) Type of the host. Argument Type: String
`pools`	(Optional) 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. Argument Type: String

Arguments	Descriptions
`port`	(Optional) The ElectricFlow 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 automation platform UI. 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 ElectricFlow agent. For `ssh`, the default is 22. Argument Type: Integer
`proxyCustomization`	(Optional) Customized Perl code specifying how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources. Argument Type: String
`proxyHostName`	(Optional) The name or IP address of the computer containing the ElectricFlow Agent used for a proxy resource. Argument Type: String
`proxyPort`	(Optional) The ElectricFlow agent port number for a proxy resource. See the `port` argument description for more details. Argument Type: Integer
`proxyProtocol`	(Optional) Protocol for communicating with the proxy target. Defaults to `ssh`. (This argument is not exposed in the ElectricFlow UI at this time.) Argument Type: String
`repositoryNames`	(Optional) A list of one or more repository names. Each repository name is listed on a "new line." Argument Type: String
`resourceDisabled`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1`, ElectricFlow will not start new steps on this resource. The default is `false`. Argument Type: Boolean
`shell`	(Optional) 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. Argument Type: String

Arguments	Descriptions
`stepLimit`	(Optional) 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. Argument Type: Integer
`trusted`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, the resource is *trusted*. Agents can be either *trusted* or *untrusted*: • **trusted**–The ElectricFlow server verifies the agent's identity using SSL certificate verification. • **untrusted**–The ElectricFlow server does not verify agent identity. An untrusted agent could be a security risk. Argument Type: Boolean
`useSSL`	(Optional) <*Boolean flag* - `0|1|true|false`> Use this flag to define whether SSL is used for server-agent communication, or if you need to use SSL to communicate with your Active Directory servers. The default is `1` or `true`. **Note:** Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server. Argument Type: Boolean
`workspaceName`	(Optional) Name of the workspace that the resource uses. Argument Type: String
`zoneName`	(Optional) The name of the zone where this resource resides. Argument Type: String

Positional arguments

`resourceName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->createResource(<resourceName>, {<optionals>});`

Example

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

ectool

syntax: `ectool createResource <resourceName> [optionals]`

Example

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

Back to Top

createResourcePool

Creates a new pool for resources.

You must specify a `resourcePoolName`.

Arguments	**Descriptions**
`resourcePoolName`	Name for the resource pool that must be unique among all resource pools. Argument Type: String
`autoDelete`	(Optional) <*Boolean flag* - `0|1|true|false`> – If this is set to `1` or `true`, the resource pool is deleted automatically when the last resource is removed from the pool. Argument Type: Boolean
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument Type: String
`orderingFilter`	(Optional) 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. Argument Type: String
`resourceNames`	(Optional) 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. Argument Type: Collection

Arguments	Descriptions
`resourcePoolDisabled`	(Optional) <*Boolean flag* - `0|1|true|false`> –If this is set to `1` or `true`, any runnable steps that refer to the pool will block until the pool is enabled again. Argument Type: Boolean

Positional arguments

`resourcePoolName`

Response

Returns a `resourcePool` object.

ec-perl

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

Example

```
$cmdr->createResourcePool ("Test Machines", {resourceName => ["Machine1", "Machine
2"]});
```

ectool

syntax: `ectool createResourcePool <resourcePoolName> [optionals]`

Example

```
ectool createResourcePool "Test Machines" --resourceNames "Machine1" "Machine2
```

Back to Top

deleteResource

Deletes a resource.

You must enter a `resourceName`.

Arguments	Descriptions
`resourceName`	Name of the resource that must be unique among all resources. Argument Type: String

Positional arguments

`resourceName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteResource("Tester 1");
```

ectool

syntax: `ectool deleteResource <resourceName>`

Example

```
ectool deleteResource "Tester 1"
```

Back to Top

deleteResourcePool

Deletes a resource pool.

You must enter a `resourcePoolName`.

Arguments	Descriptions
`resourcePoolName`	Name for the resource pool that must be unique among all resource pools. A resource pool contains one or more resources. Argument Type: String

Positional arguments

`resourcePoolName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool deleteResourcePool <resourcePoolName>`

Example

```
ectool deleteResourcePool "Test Suite 1"
```

Back to Top

getResource

Retrieves a resource by its name.

You must specify `resourceName`.

Arguments	Descriptions
`resourceName`	Name for the resource that must be unique among all resources. Argument Type: String

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 so on. 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

Retrieves the list of resources in an environment tier.

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

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

Positional arguments

`projectName, environmentName, environmentTierName`

Response

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

ec-perl

Syntax:

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

Example:

```
$ec->getResourcesInEnvironmentTier("Default", "PROD", "Tomcat");
```

ectool

Syntax:

```
getResourcesInEnvironmentTier <projectName> <environmentName> <environmentTierNa
me>
```

Example:

```
ectool getResourcesInEnvironmentTier "Default" "PROD" "Tomcat"
```

Back to Top

getResourcesInPool

Retrieves a list of resources in a pool.

You must specify a `resourcePoolName`.

Arguments	Descriptions
`resourcePoolName`	The name of a resource pool containing one or more resources. Argument Type: String
`jobStepId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job step that is assigned automatically when the job step is created. This is UUID of the job step related to this pool. Argument type: UUID

Positional arguments

`resourcePoolName`

Response

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

ec-perl

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

Example

```
$cmdr->getResourcesInPool("Windows Pool");
```

ectool

syntax: `ectool getResourcesInPool <resourcePoolName> [optionals]`

Example

```
ectool getResourcesInPool "Windows Pool"
```

Back to Top

getResourcePool

Retrieves a specified resource pool by name.

You must specify a `resourcePoolName`.

Arguments	Descriptions
`resourcePoolName`	Name for the resource pool that must be unique among all resource pools. Argument Type: String

Positional arguments

`resourcePoolName`

Response

An XML stream containing one `resourcePool` element.

ec-perl

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

Example

```
$cmdr->getResourcePool("Windows Pool");
```

ectool

syntax: `ectool getResourcePool <resourcePoolName>`

Example

```
ectool getResourcePool "Windows Pool"
```

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: For a proxy resource, the `proxyHostName` and `proxyPort` arguments refer to the proxying ElectricFlow agent. The `hostName` and `port` arguments refer to the proxy target.

Arguments	Descriptions
`resourceName`	Name for the resource that must be unique among all resources. Argument Type: String
`artifactCacheDirectory`	(Optional) The directory on the agent host where retrieved artifacts are stored. Argument Type: String
`block`	(Optional) <*Boolean flag -* `0|1|true|false`> A newly modified resource will be pinged. This argument makes the `modifyResource` call "block" until the result of the ping is known. The default is "false". Argument Type: Boolean
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument Type: String
`hostName`	(Optional) The name or IP address for the ElectricFlow machine containing the agent for this resource. Argument Type: String
`hostType`	(Optional) The type of the host. Argument Type: String
`newName`	(Optional) New name of an existing resource. Argument Type: String
`pools`	(Optional) 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. ElectricFlow chooses a resource from the pool when it executes the job step. Argument Type: String

Arguments	Descriptions
`port`	(Optional) The port number for the ElectricFlow 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. Argument Type: Integer
`proxyCustomization`	(Optional) Customized Perl code specifying how the proxy resource communicates with the proxy target. This applies only to proxy resources. Argument Type: String
`proxyHostName`	(Optional) The IP address of the computer containing the ElectricFlow Agent used for a proxy resource. Argument Type: String
`proxyPort`	(Optional) The ElectricFlow agent port number for a proxy resource. See the `port` argument for more details. Argument Type: Integer
`proxyProtocol`	(Optional) Protocol for communicating with the proxy target. Defaults to `ssh`. This argument is not exposed in the ElectricFlow UI at this time. Argument Type: String
`repositoryNames`	(Optional) A list of repository names with each repository name listed on a "new line". Argument Type: String
`resourceDisabled`	<*Boolean flag -* `0|1|true|false`> If this is set to `1`or `true`, ElectricFlow will not start new steps on this resource. Argument Type: String
`shell`	(Optional) 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. Argument Type: String
`stepLimit`	(Optional) 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. Argument Type: String

Arguments	Descriptions
`trusted`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `1` or `true`, the resource is *trusted*. Agents can be either *trusted* or *untrusted*: • trusted–The ElectricFlow server verifies the agent's identity using SSL certificate verification. • untrusted–The ElectricFlow server does not verify agent identity. An untrusted agent could be a security risk. Argument Type: Boolean
`useSSL`	(Optional) <*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. The default is `1` or `true`. **Note:** Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server. Argument Type: Boolean
`workspaceName`	(Optional) Name of the default workspace where job output is stored. Argument Type: String
`zoneName`	(Optional) Name of the zone where this resource resides, which must be unique among all zones. Argument Type: String

Positional arguments

`resourceName`

Response

Returns a modified resource object.

ec-perl

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

Example

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

ectool

syntax: `ectool modifyResource <resourceName> [optionals]`

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
`resourcePoolName`	Name for the resource pool that must be unique among all resource pools. A resource pool contains one or more resources. Argument Type: String
`autoDelete`	(Optional) <*Boolean flag -* `0|1|true|false`> If this is set to `1` or `true`, the resource pool will be deleted automatically when the last resource is removed form the pool. Argument Type: Boolean
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument Type: String
`newName`	(Optional) Any new unique name for this resource pool. Argument Type: String
`orderingFilter`	(Optional) 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. Argument Type: String
`resourceNames`	(Optional) 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. Argument Type: Collection
`resourcePoolDisabled`	(Optional) <*Boolean flag -* `0|1|true|false`> If this is set `1` or `true`, any runnable steps that refer to the pool will block until the pool is enabled again. Argument Type: Boolean

Positional arguments

`resourcePoolName`

Response

Returns the modified `resourcePool` object.

ec-perl

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

Example

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

ectool

syntax: `ectool modifyResourcePool <resourcePoolName> [optionals]`

Example

```
ectool modifyResourcePool "Windows Pool" --resourcePoolDisabled 1
```

Back to Top

pingAllResources

Pings all resources.

Arguments	Description
`block`	(Optional) <*Boolean flag* - `0|1|true|false`> The 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). Argument Type: Boolean

Positional arguments

None

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool pingAllResources [optionals]`

Example

```
ectool pingAllResources
```

Back to Top

pingResource

Pings the specified resource.

You must specify the `resourceName`.

Arguments	Descriptions
`resourceName`	Name for the resource that must be unique among all resources. Argument Type: String

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 a resource from the specified environment tier.

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

Arguments	Descriptions
`resourceName`	Name for the resource that must be unique among all resources. Argument Type: String

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

Response

Removes the resource from an environment tier.

ec-perl

Syntax:

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

Example:

```
$ec->removeResourceFromEnvironmentTier("Web server", "Default", "QA", "Tomcat");
```

ectool

Syntax:

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

Example:

```
ectool removeResourceFromEnvironmentTier "Web server" "Default" "QA" "Tomcat"
```

Back to Top

removeResourcesFromPool

Removes resources from the specified resource pool.

You must specify a `resourcePoolName`.

Arguments	Descriptions
`resourcePoolName`	Name for the resource pool that must be unique among all resource pools. Argument Type: String

Arguments	Descriptions
`resourceNames`	(Optional) The list of resources to remove from this pool. Argument Type: Collection

Positional arguments

`resourcePoolName`

Response

Removes the selected resources from the resource pool.

ec-perl

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

Example

```
$ec->removeResourcesFromPool("Test machines", {resourceNames => ["Tester 1", "Teste
r 2", "Tester3"]});
```

ectool

syntax: `ectool removeResourcesFromPool <resourcePoolName> (optionals)`

Example

```
ectool removeResourcesFromPool "Test machines" --resourceNames "Tester 1" "Tester 2
" "Tester 3"
```

Back to Top

runDiscovery

Runs the Discover procedure in a plugin to discover contents of a list or set of resources, and store settings for them in the `ec_discovery` property sheets.

You must specify the `pluginKey`, `pluginName`, `projectName`, `environmentName`, `environmentTierName`, `configurationName`.

Arguments	Descriptions
`pluginKey`	The version independent name of the plugin. Argument type: String
`pluginName`	The name of the plugin. Argument type: String
`projectName`	The name of the project. Argument type: String

Arguments	Descriptions
`environmentName`	The name of the environment. Argument type: String
`environmentTierName`	The name of the environment tier. Argument type: String
`configurationName`	The name of the plugin configuration. Argument type: String
`resourceNames`	(Optional) List of resources to use for discovery.. Argument type: Collection

Positional arguments

`pluginKey`, `pluginName`, `projectName`, `environmentName`, `environmentTierName`, `configurationName`

Response

None or a status OK message.

ec-perl

syntax: `$<object>->runDiscovery(<pluginKey>, <pluginName>, <projectName>, <environmentName>, <environmentTierName>, <configurationName>, {<optionals>});`

Example

```
$ec->runDiscovery("EC-OpenStack", "OpenStack", "Default", "PROD", "DB server", "SQL
config",
{resourceNames => DB1 DB3 DB7});
```

ectool

syntax: `ectool runDiscovery <pluginKey> <pluginName> <projectName> <environmentName> <environmentTierName> <configurationName> [optionals]`

Example

```
ectool runDiscovery "EC-OpenStack" "OpenStack" "Default" "PROD" "DB server" "SQLcon
fig" --resourceNames DB1 DB3 DB7}
```

Back to Top

signCertificate

Signs an agent certificate.

Arguments	Descriptions
`certificateData`	A Privacy Enhanced Mail (PEM) encoded certificate signing request. Argument Type: String

Positional arguments

`certificateData`

Response

None or a status OK message.

ec-perl

syntax:`$cmdr->signCertificate (<certificateData>);`

Example

```
$cmdr->signCertificate ("MIIEczCCA1ugAwIBAgIBADANBgkqhkiG9w0BAQQFAD");
```

ectool

syntax:`ectool signCertificate <certificateData>`

Example

```
ectool signCertificate "MIIEczCCA1ugAwIBAgIBADANBgkqhkiG9w0BAQQFAD"
```

Back to Top

API Commands - Rolling Deploy

createRollingDeployPhase

Adds a rolling deploy phase to the specified environment.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`rollingDeployPhaseName`	Name of the rolling deploy phase. Argument Type: String
`description`	(Optional) Comment text describing this object. It is not interpreted by ElectricFlow. Argument Type: String
`environmentName`	(Optional) Name of the environment. Argument Type: String
`orderIndex`	(Optional) Order of the phases at runtime, starting from 1. Argument Type: Integer
`phaseExpression`	(Optional) This is fixed text or text embedding a 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. Argument Type: String
`rollingDeployPhaseType`	(Optional) Type of rolling deploy phase. Valid values are `tagged` and `untagged`. Argument Type: RollingDeployPhaseType

Positional arguments

`projectName` and `rollingDeployPhaseName`

Response

Returns a rolling-deploy phase object.

ec-perl

syntax: `$<object>->createRollingDeployPhase(<projectName>, <rollingDeployPhaseName>, {<optionals>});`

Example

```
$ec->createRollingDeployPhase('Default', 'Test Phase', {environmentName => 'Q
A'});
```

ectool

syntax: `ectool createRollingDeployPhase <projectName> <rollingDeployPhaseName> [optionals]`

Example

```
ectool createRollingDeployPhase 'Default' 'Test Phase' --environmentName 'QA'
```

Back to Top

deleteRollingDeployPhase

Deletes the rolling deploy phase associated with an environment.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`rollingDeployPhaseName`	Name of the rolling deploy phase. Argument Type: String
`environmentName`	(Optional) Name of the environment. Argument Type: String

Positional arguments

`projectName` and `rollingDeployPhaseName`

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->deleteRollingDeployPhase(<projectName>, <rollingDeployPhaseName>, {<o
ptionals>});
```

Example:

```
$ec->deleteRollingDeployPhase('Default', 'Test Phase', {environmentName => 'Q
A'});
```

ectool

Syntax:

```
ectool deleteRollingDeployPhase <projectName> <rollingDeployPhaseName> [optional
s]
```

Example:

```
ectool deleteRollingDeployPhase 'Default' 'Test Phase' --environmentName 'QA'
```

Back to Top

getRollingDeployPhase

Retrieves the rolling deploy phase associated with an environment.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`rollingDeployPhaseName`	Name of the rolling deploy phase. Argument Type: String
`environmentName`	(Optional) Name of the environment. Argument Type: String
`environmentTierName`	(Optional) Name for the environment tier. Argument Type: String
`includeResourceCount`	(Optional) <*Boolean flag -* `0|1|true|false`> If this is set to `true` or `1`, the response includes the resource count for the rolling deploy phase. Argument Type: Boolean

Positional arguments

`projectName` and `rollingDeployPhaseName`

Response

Returns a rolling deploy phase object.

ec-perl

syntax: `$<object>->getRollingDeployPhase(<projectName>, <rollingDeployPhaseName>, {<optionals>});`

Example

```
$ec->getRollingDeployPhase('Default', 'Test Phase', {environmentName => 'QA', in
cludeResourceCount => true});
```

ectool

syntax: `ectool getRollingDeployPhase <projectName> <rollingDeployPhaseName> [optionals]`

Example

```
ectool getRollingDeployPhase 'Default' 'Test Phase' --environmentName 'QA' --inc
ludeResourceCount true
```

Back to Top

getRollingDeployPhases

Retrieves all the rolling deploy phases associated with an environment.

You must specify the `projectName`.

Arguments	Descriptions
`applicationName`	(Optional) Name for the application. Argument Type: String
`applicationProjectName`	(Optional) Name for the application project. Argument Type: String
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`environmentName`	(Optional) Name of the environment. Argument Type: String
`environmentTierName`	(Optional) Name for the environment tier. Argument Type: String
`includeResourceCount`	(Optional) <*Boolean flag -* `0|1|true|false`> If this is set to `true` or `1`, the response includes the resource count for the rolling deploy phase. Argument Type: Boolean

Positional arguments

`projectName`

Response

Returns all the rolling deploy phase objects associated with an environment.

ec-perl

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

Example

```
$ec->getRollingDeployPhases('Default', {environmentName => 'QA', includeResource
Count => true});
```

ectool

syntax: `ectool getRollingDeployPhases <projectName> [optionals]`

Example

```
ectool getRollingDeployPhases 'Default' --environmentName 'QA' --includeResource
Count true
```

Back to Top

modifyRollingDeployPhase

Modifies the rolling deploy phase associated with an environment.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument Type: String
`rollingDeployPhaseName`	Name of the rolling deploy phase. Argument Type: String
`description`	(Optional) Comment text describing this object. It is not interpreted by ElectricFlow. Argument Type: String
`environmentName`	(Optional) Name of the environment. Argument Type: String
`newName`	(Optional) New name of the rolling deploy phase. Argument Type: String
`orderIndex`	(Optional) Order of the phases at runtime, starting from 1. Argument Type: Integer
`phaseExpression`	(Optional) This is fixed text or text embedding a 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. Argument Type: String
`rollingDeployPhaseType`	(Optional) Type of rolling deploy phase. Valid values are `tagged` and `untagged`. Argument Type: RollingDeployPhaseType

Positional arguments

`projectName` and `rollingDeployPhaseName`

Response

Returns a modified rolling deploy phase object.

ec-perl

syntax: `$<object>->modifyRollingDeployPhase(<projectName>, <rollingDeployPhaseName>, {<optionals>});`

Example

```
$ec->modifyRollingDeployPhase('Default', 'Test Phase', {newName => 'QA Test Phas
e', orderIndex => 3});
```

ectool

syntax: `ectool modifyRollingDeployPhase <projectName> <rollingDeployPhaseName> [optionals]`

Example

```
ectool modifyRollingDeployPhase 'Default' 'Test Phase' --newName 'QA Test Phase'
--orderIndex 3
```

Back to Top

setTierResourcePhase

Maps a resource to a rolling deploy phase.

Required Arguments

`projectName`

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

Argument Type: String

`environmentName`

Description: The name of the environment.

Argument Type: String

`environmentTierName`

Description: Name for the environment tier that must be unique among all tiers for the environment.

Argument Type: String

Optional Arguments

`resourceNames`

Description: A list of resources to be mapped to a rolling deploy phase.

Argument Type: Collection

`resourcePhaseMappings`

Description: A map of resources to rolling deploy phases.

Argument Type: Map

`resourcePoolNames`

Description: A list of resource pools to be mapped to a rolling deploy phase.

Argument Type: Collection

`resourcePoolPhaseMappings`

Description: A map of resource pools to rolling deploy phases.

Argument Type: Map

`rollingDeployPhaseName`

Description: Name for the rolling deploy phase to be associated with resources or resource pools.

Argument Type: String

Response

Returns zero or more mappings of resources or resource pools to rolling deploy phases.

ec-perl

Syntax:

```
$<object>->setTierResourcePhase(<projectName>, <environmentName>, <environmentTi
erName>, {<optionals>});
```

Examples:

To map three resources to a rolling deploy phase:

```
$ec->setTierResourcePhase("Default", "PROD", "Web servers", {resourceNames => We
bserver-resource1
Webserver-resource2 Webserver-resource3, rollingDeployPhaseName => Phase1});
```

To map three resource pools to a rolling deploy phase:

```
$ec->setTierResourcePhase("Default", "PROD", "Web servers", {resourceNames => We
bserver-pool1
Webserver-pool2 Webserver-pool3, rollingDeployPhaseName => Phase1});
```

To map specific resources to specific rolling deploy phases:

```
$ec->setTierResourcePhase("Default", "PROD", "Web servers", {resourcePhaseMappin
gs => Webserver-resource1=Phase1
Webserver-resource2=Phase2 Webserver-resource3=Phase1});
```

To map specific resource pools to specific rolling deploy phases:

```
$ec->setTierResourcePhase("Default", "PROD", "Web servers", {resourcePhaseMappin
gs => Webserver-pool1=Phase1
Webserver-pool2=Phase2 Webserver-pool3=Phase1});
```

ectool

Syntax:

```
ectool setTierResourcePhase <projectName> <environmentName> <environmentTierNam
e> [optionals]
```

Examples:

To map three resources to a rolling deploy phase:

```
ectool setTierResourcePhase "Default" "PROD" "Web servers" --resourceNames Webse
rver-resource1 Webserver-resource2
Webserver-resource3 --rollingDeployPhaseName Phase1
```

To map three resource pools to a rolling deploy phase:

```
ectool setTierResourcePhase "Default" "PROD" "Web servers" --resourcePoolNames W
ebserver-pool1 Webserver-pool2
Webserver-Pool3 --rollingDeployPhaseName Phase1
```

To map specific resources to specific rolling deploy phases:

```
ectool setTierResourcePhase "Default" "PROD" "Web servers" --resourcePhaseMappin
gs Webserver-resource1=Phase1
Webserver-resource2=Phase2 Webserver-resource3=Phase1
```

To map specific resource pools to specific rolling deploy phases:

```
ectool setTierResourcePhase "Default" "PROD" "Web servers" --resourcePhaseMappin
gs Webserver-pool1=Phase1
Webserver-pool2=Phase2 Webserver-pool3=Phase1
```

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
projectName	Name for the project that must be unique among all projects. Argument type: String
scheduleName	Name for the schedule that must be unique among all schedules for the project. Argument type: String

Arguments	Descriptions
`actualParameters`	(Optional) 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. Argument type: Map
`applicationName`	(Optional) The name of the application that owns the process. Argument type: String
`applicationProjectName`	(Optional) The name of the project to which the application belongs. Argument type: String
`beginDate`	(Optional) <*yyyy-mm-dd*> The date you want the schedule to begin. Argument type: String
`credentialName`	(Optional) 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. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`endDate`	(Optional) <*yyyy-mm-dd*> The date you want this schedule to end. Argument type: String
`environmentName`	(Optional) The name of the environment where the application runs. Argument type: String

Arguments	**Descriptions**
`environmentProjectName`	(Optional) The name of the project to which the environment or environment template belongs. Argument type: String
`environmentTemplateName`	(Optional) The name of the environment template used to create the dynamic environment. Argument type: String
`environmentTemplateProjectName`	(Optional) Name of the project containing specified environment template. Argument Type: String
`environmentTemplateTierMapName`	(Optional) The name of the environment template tier map used to create the dynamic environment where the application runs. Argument type: String
`insertRollingDeployManualStep`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` a manual step needs to be added after each phase or batch is run. Argument type: Boolean
`interval`	(Optional) Determines the repeat interval for starting new jobs. If specified, the procedure, process, or workflow will be rescheduled continuously at intervals of this length. "Continuous" means that the procedure, process, or workflow is rescheduled as soon as the previous job finishes. Argument type: String
`intervalUnits`	(Optional) Specifies the units for the `interval` argument `<hours|minutes|seconds|continuous>` If set to `continuous`, ElectricFlow creates a new job as soon as the previous job completes. Argument type: String
`misfirePolicy`	(Optional) `<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". Argument type: MisfirePolicy

Arguments	**Descriptions**
`monthDays`	(Optional) Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space. Argument type: String
`pipelineName`	(Optional) Name of the pipeline to run. Argument type: String
`priority`	(Optional) `<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. Argument type: JobPriority
`procedureName`	(Optional) Name of the procedure to run when the schedule is invoked. Argument type: String
`processName`	(Optional) Name of the application process to run when the schedule is invoked. Argument type: String
`releaseName`	(Optional) The name of the release to run when the schedule is invoked. Argument type: String
`rollingDeployEnabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` the pipeline runs the rolling deployment. Argument type: Boolean
`rollingDeployManualStepAssignees`	(Optional) A list of assignees who receive the notification when the rolling deploy iteration is completed. Argument type: Collection
`rollingDeployManualStepCondition`	(Optional) The **Run if** condition (run condition) on the manual step that was created during the rolling deployment. Argument type: NotificationType

Arguments	Descriptions
`rollingDeployPhases`	(Optional) One or more rolling deploy phases to be used in the rolling deployment. Argument type: Collection
`scheduleDisabled`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to 1, ElectricFlow will not start any new jobs from the schedule. Defaults to "false". Argument type: Boolean
`snapshotName`	(Optional) Name of the snapshot used to invoke the application process. Argument type: String
`stagesToRun`	(Optional) One or more stages to run in a pipeline. Argument type: Collection
`startTime`	(Optional) The time of day to run the procedure, process, or workflow when the schedule is invoked. Using this schedule, ElectricFlow starts creating jobs at this time on the specified days. Enter hours and minutes, formatted `hh:mm`, using the 24-hour clock (for example, 17:00). Argument type: String
`startingStage`	(Optional) Name of the stage where the pipeline starts. Argument type: String
`startingStateName`	(Optional) Name of the starting state of the workflow. Argument type: String
`stopTime`	(Optional) The time of day to stop invoking the schedule. ElectricFlow stops creating new jobs at this time, but a job in progress continues to run. If `stopTime` is not specified, ElectricFlow creates one job only on each specified day. Enter hours and minutes, formatted `hh:mm`, using the 24-hour clock (for example, 17:00). Argument type: String
`tierMapName`	(Optional) Name of the tier map that determines in which environment the application runs. Argument type: String
`tierResourceCounts`	Resource count for each resource template tier. Argument type: Map

Arguments	Descriptions
`timeZone`	(Optional) Enter the time zone (string) you want to use for this schedule. Argument type: String
`weekDays`	(Optional) 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. Argument type: String
`workflowName`	(Optional) Name of the workflow to run when the schedule is invoked. Argument type: String

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> [optional]`

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 project to which the schedule belongs. Argument type: String
`scheduleName`	The name of the schedule. Argument type: String

Positional arguments

`projectName, scheduleName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteSchedule("Default", "Weekend Tests");
```

ectool

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

Example

```
ectool deleteSchedule "Default" "Weekend Tests"
```

Back to Top

getSchedule

Retrieves a schedule by its name.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`scheduleName`	Name for the schedule that must be unique among all schedules for the project. Argument type: String

Positional arguments

`projectName, scheduleName`

Response

One `schedule` element.

ec-perl

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

Example

```
$cmdr->getSchedule("Default", "Weekend Tests");
```

ectool

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

Example

```
ectool getSchedule "Default" "Weekend Tests"
```

Back to Top

getSchedules

Retrieves all schedules.

You must specify a `projectName`.

Arguments	Descriptions
`projectName`	The name for the project that must be unique among all projects. Argument type: String
`applicationName`	(Optional) The name of the application that owns the process. Argument type: String
`applicationProjectName`	(Optional) The name for the project to which the application belongs. Argument type: String
`includeWorkflows`	(Optional) To include workflow related schedules. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`processName`	(Optional) The name of the application process. Argument type: String
`releaseName`	(Optional) The name of the release. Argument type: String

Arguments	Descriptions
`tierMapName`	(Optional) The name of the tier map that determines the environment in which run the process. Argument type: String

Positional arguments

`projectName`

Response

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

ec-perl

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

Example

```
$cmdr->getSchedules("Default" {applicationName => "Deploy"});
```

ectool

syntax: `ectool getSchedules <projectName> [optional]`

Example

```
ectool getSchedules "Default" --applicationName "Deploy"
```

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
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`scheduleName`	The name of the schedule that must be unique among all schedules for the project. Argument type: String

Arguments	Descriptions
`actualParameters`	(Optional) 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. Argument type: Map
`applicationProjectName`	(Optional) Name of the project to which the application belongs. Argument type: String
`beginDate`	(Optional) <*yyyy-mm-dd*> The date you want the schedule to begin (for example, 2016-05-15). Argument type: String
`clearActualParameters`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `true` or `1`, all the actual parameters are removed from the schedule. Argument type: Boolean
`credentialName`	(Optional) 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. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`endDate`	(Optional) <*yyyy-mm-dd*> The date you want this schedule to end (for example, 2016-05-15). The end date is not included in the range of dates. Argument type: String

Arguments	Descriptions
environmentName	(Optional) The name of the environment where the process will run. Argument type: String
environmentProjectName	(Optional) The name of the project to which the environment or environment template belongs. Argument type: String
environmentTemplateName	(Optional) The name of the environment template used to determine where the process will run. Argument type: String
environmentTemplateProjectNa me	(Optional) Name of the project containing specified environment template. If this argument is not specified, the default is the environment project name. Argument Type: String
environmentTemplateTierMapNa me	(Optional) The name of the environment template tier map used to determine how to spin up the environment where the process will run. Argument type: String
insertRollingDeployManualSte p	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` a manual step needs to be added after each phase or batch is run. Argument type: Boolean
interval	(Optional) Determines the repeat interval for starting new jobs. Argument type: String
intervalUnits	(Optional) Specifies the units for the `interval` argument `<hours|minutes|seconds|continuous>`. If set to `continuous`, ElectricFlow creates a new job as soon as the previous job completes. Argument type: IntervalUnits
misfirePolicy	(Optional) `<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. The default is `ignore`. Argument type: IntervalUnits

Arguments	Descriptions
`monthDays`	(Optional) Restricts the schedule to specified days of the month. Specify numbers from 1 to 31, separating multiple numbers with a space. Argument type: String
`newName`	(Optional) New name of the schedule. Argument type: String
`pipelineName`	(Optional) Name of the pipeline to run. Argument type: String
`priority`	(Optional) `<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. Argument type: JobPriority
`procedureName`	(Optional) The name of the procedure to run when the schedule is invoked. Argument type: String
`processName`	(Optional) The name of the process to run when the schedule is invoked. Argument type: String
`releaseName`	(Optional) The name of the release to invoke. Argument type: String
`rollingDeployEnabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1` the pipeline runs the rolling deployment. Argument type: Boolean
`rollingDeployManualStepAssignees`	(Optional) A list of assignees who receive the notification when the rolling deploy iteration is completed. Argument type: Collection
`rollingDeployManualStepCondition`	(Optional) The **Run if** condition (run condition) on the manual step that was created during the rolling deployment. Argument type: NotificationType

Arguments	Descriptions
`rollingDeployPhases`	(Optional) One or more rolling deploy phases to be used in the rolling deployment. Argument type: Collection
`scheduleDisabled`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1`, ElectricFlow does not start any new jobs from the schedule. Argument type: Boolean
`snapshotName`	(Optional) The name of the snapshot to use when the application process is invoked. Argument type: String
`startTime`	(Optional) The time of day to begin running the procedure, process, or workflow when the schedule is invoked. Enter hours and minutes, formatted `hh:mm`, using the 24-hour clock (for example, 17:00). ElectricFlow starts creating jobs at this time on the days specified. Argument type: String
`stagesToRun`	(Optional) One or more stages to run in a pipeline. Argument type: Collection
`startingStage`	(Optional) Name of the stage where the pipeline starts. Argument type: String
`startingStateName`	(Optional) The name of the starting state of the workflow. Argument type: String
`stopTime`	(Optional) The time of day to stop invoking the schedule. ElectricFlow stops creating new jobs at this time, but a job in progress continues to run. If `stopTime` is not specified, ElectricFlow creates one job only on each specified day. Enter hours and minutes, formatted `hh:mm`, using the 24-hour clock (for example, 17:00). Argument type: String
`tierMapName`	(Optional) The name of the tier map used to determine where to run the process. It maps application processes to environments. Argument type: String
`tierResourceCounts`	(Optional) The resource count for each resource template tier. Argument type: String

Arguments	Descriptions
`timeZone`	(Optional) Enter the time zone you want to use for this schedule. Argument type: String
`weekDays`	(Optional) 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. Argument type: String
`workflowName`	(Optional) The name of the workflow to run when the schedule is invoked. Argument type: String

Positional arguments

`projectName, scheduleName`

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: `ectool modifySchedule <projectName> <scheduleName> [optionals]`

Example

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

Back to Top

pauseScheduler

Sets the scheduler to pause.

Arguments	Descriptions
`paused`	<*Boolean flag* - `0|1|true|false`> If this argument set to `1` or `true`, set the scheduler to pause. If this argument is set to `0` or `false`, set the scheduler to not pause.

Positional arguments

`paused`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->pauseScheduler (<paused>);`

Example

```
$cmdr->pauseScheduler (true);
```

ectool

syntax: `ectool pauseScheduler <paused>`

Example

```
ectool pauseScheduler true
```

Back to Top

API Commands - Search Filters

createSearchFilter

Creates a search filter.

Required Arguments

`objectType`

Description: The object type. Values include:

```
application          process
artifact             processStep
artifactVersion      project
component            property
credential           repository
directoryProvider    resource
emailconfig          resourcePool
emailNotifier        schedule
environment          state
formalParameter      stateDefinition
job                  transition
jobStep              transitionDefinition
logEntry             workflow
plugin               workflowDefinition
procedure            workspace
procedureStep
```

Argument Type: String

`searchFilterName`

Description: Name of the search filter.

Argument Type: String

Optional Arguments

`filters`

Description: Filters to use during the search operation.

Argument Type: Collection

`quickSearchFilter`

Description: Quick search filter criteria.

Argument Type: String

Response

Returns a search filter object.

ec-perl

Syntax:

```
$<object>->createSearchFilter(<objectType>, <searchFilterName>, {<optionals>});
```

Example:

```
$ec->createSearchFilter("workflows", "Daily QA");
```

ectool

Syntax:

```
ectool createSearchFilter <objectType> <searchFilterName> [optionals]
```

Example:

```
ectool createSearchFilter "workflows" "Daily QA"
```

Back to Top

deleteSearchFilter

Delete a search filter from the database.

Required Arguments

`objectType`

Description: The object type. See createSearchFilter on page 641 for a list of possible values.

Argument Type: String

`searchFilterName`

Description: Name of the search filter.

Argument Type: String

Optional Arguments

None

Response

None or a status OK message.

ec-perl

Syntax:

```
$<object>->deleteSearchFilter(<objectType>, <searchFilterName>});
```

Example:

```
$ec->deleteSearchFilter("workflows", "Daily QA");
```

ectool

Syntax:

```
ectool deleteSearchFilter <objectType> <searchFilterName>
```

Example:

```
ectool deleteSearchFilter "workflows" "Daily QA"
```

Back to Top

getSearchFilter

Retrieves the specified search filter.

Required Arguments

`objectType`

Description: The object type. See createSearchFilter on page 641 for a list of possible values.

Argument Type: String

`searchFilterName`

Description: Name of the search filter.

Argument Type: String

Optional Arguments

None

Response

Returns the specified search filter.

ec-perl

Syntax:

```
$<object>->getSearchFilter(<objectType>, <searchFilterName>});
```

Example:

```
$ec->getSearchFilter("workflows", "Daily QA");
```

ectool

Syntax:

```
ectool getSearchFilter <objectType> <searchFilterName>
```

Example:

```
ectool getSearchFilter "workflows" "Daily QA"
```

Back to Top

getSearchFilters

Retrieves all the search filters for an object type.

Required Arguments

`objectType`

Description: The object type. See createSearchFilter on page 641 for a list of possible values.

Argument Type: String

Optional Arguments

None

Response

Returns one or more search filters.

ec-perl

Syntax:

```
$<object>->getSearchFilters(<objectType>});
```

Example:

```
$ec->getSearchFilters("processStep");
```

ectool

Syntax:

```
ectool getSearchFilters <objectType>
```

Example:

```
ectool getSearchFilters "processStep"
```

Back to Top

modifySearchFilter

Modifies a search filter.

Required Arguments

`objectType`

Description: The object type. See createSearchFilter on page 641 for a list of possible values.

Argument Type: String

`searchFilterName`

Description: Name of the search filter.

Argument Type: String

Optional Arguments

`filters`

Description: Filters to use during the search operation.

Argument Type: Collection

`quickSearchFilter`

Description: Quick search filter criteria.

Argument Type: String

Response

Returns a modified search filter object.

ec-perl

Syntax:

```
$<object>->modifySearchFilter(<objectType>, <searchFilterName>, {<optionals>});
```

Example:

```
$ec->modifySearchFilter("workflows", "Daily QA", {filters => Filter1 Filter2 Fil
ter3);
```

ectool

Syntax:

```
ectool modifySearchFilter <objectType> <searchFilterName> [optionals]
```

Example:

```
ectool modifySearchFilter "workflows" "Daily QA" --filters Filter1 Filter2 Filte
r3
```

Back to Top

API Commands - Server Management

deleteLicense

Deletes a license.

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

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

Positional arguments

`productName, featureName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteLicense("ElectricFlow", "Server");
```

ectool

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

Example

```
ectool deleteLicense "ElectricFlow" "Server"
```

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

getCertificates

Returns the certificates in the trust chain for the server.

Arguments	Descriptions
None	–

Positional arguments

None.

Response

None or a status OK message.

ec-perl

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

Examples

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

ectool

syntax:`ectool getCertificates`

Examples

```
ectool getCertificates
```

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` Argument: String
`productName`	The name of the product with the licensed feature. Possible products include: `ElectricFlow` Argument: String

Positional arguments

`productName, featureName`

Response

One `license` element.

ec-perl

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

Example

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

ectool

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

Example

```
ectool getLicense "ElectricFlow" "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

getServerInfo

Retrieves information about server ports and message delivery.

Arguments	Descriptions
None	–

Positional arguments

None.

Response

Returns the information about server ports and message delivery.

ec-perl

syntax:`$cmdr->getServerInfo;`

Examples

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

ectool

syntax:`ectool getServerInfo`

Examples

```
ectool getServerInfo
```

Back to Top

getServerStatus

Retrieves the current status of the ElectricFlow server, including log messages generated during the startup sequence.

Arguments	Descriptions
`block`	(Optional) <*Boolean flag* - `0|1|true|false`> If the argument is set to `1` or `true`, the system waits until the server reaches a *terminal state*. Terminal states include `running`, `failed`, `stopping`, and `importFailed`. Argument: Boolean
`diagnostics`	(Optional) <*Boolean flag* - `0|1|true|false`> Select the diagnostic information that you want 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. Argument: Boolean

Arguments	Descriptions
`serverStateOnly`	(Optional)<*Boolean flag* - `0|1|true|false`> If the argument is set to `1` or `true`, the system limits the response to the short form and causes ectool to return only the value of the `serverStatus` element as a simple string value. Argument: Boolean
`timeout`	(Optional) This argument specifies the timeout for the `element` flag. The default value is 120 seconds. Argument: Integer

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

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

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

logMessage

Enters a message in the server log.

Arguments	Descriptions
`message`	Message to add to the server log. Argument: String
`level`	(Optional) Select a message level: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, or `OFF`. Argument: String
`logger`	(Optional) Name of the object that logged the message. Argument: String

Positional arguments

`message`

Response

None or a status OK message.

ec-perl

syntax:`$cmdr->logMessage (<message>, <optionals>});`

Examples

```
$cmdr->logMessage ("abort job step" {level => INFO});
```

ectool

syntax:`ectool logMessage <message> [optionals]`

Examples

```
ectool logMessage "abort job step" --level INFO
```

Back to Top

setLogLevel

Changes the log level of a logger.

Arguments	Descriptions
`logger`	Name of the user or resource that logged the message. Argument: String
`level`	Select a message level: `TRACE`, `DEBUG`, `INFO`, `WARN`, `ERROR`, or `OFF`. Argument: String

Positional arguments

```
logger, level
```

Response

None or a status OK message.

ec-perl

syntax:`$cmdr->setLogLevel(<logger>, <level>);`

Examples

```
$cmdr->setLogLevel ("Test Lab 1", INFO);
```

ectool

syntax:`ectool setLogLevel <logger> <level>`

Examples

```
ectool setLogLevel "Test Lab 1" INFO
```

Back to Top

shutdownServer

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

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 [optionals]`

Example

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

Back to Top

tunePerformance

Adjusts how the server is performing.

Arguments	Descriptions
`apiQueueSize`	Maximum number of threads that will be created in the API thread pool. Argument: Integer
`dbConnectionPoolSize`	Maximum number of database connections that the server will use. Argument: Integer

Arguments	Descriptions
`dbThreadPoolSize`	Number of worker threads in the database connection manager. Argument: Integer
`dispatchQueueSize`	Maximum number of threads that will be created in the Dispatch thread pool. Argument: Integer
`quartzQueueSize`	Maximum number of threads that will be created in the Quartz thread pool. Argument: Integer
`stateMachineQueueSize`	Maximum number of threads that will be created in the `stateMachine` thread pool. Argument: Integer

Positional arguments

None

Response

None or a status OK message.

ec-perl

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

Examples

```
$cmdr->tunePerformance ({apiQueueSize => 4, dbConnectionPoolSize => 2});
```

ectool

syntax:`ectool tunePerformance [optionals]`

Examples

```
ectool tunePerformance --apiQueueSize 4 --dbConnectionPoolSize 2
```

Back to Top

API Commands - Snapshots

modifySnapshot on page 663

createSnapshot

Creates a new snapshot of the specified application.

You must specify `projectName` , `applicationName`, and `snapshotName`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`applicationName`	The name of the application that must be unique among all projects. Argument type: String
`snapshotName`	The name of the snapshot that must be unique withing the application. Argument type: String
`componentVersions`	(Optional) The name and version of the component for the snapshot. Argument type: Map
`description`	(Optional) Comment text describing this object, which is not interpreted by ElectricFlow. A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`environmentName`	(Optional) Name of the environment from which the snapshot is created. Argument type: String
`environmentProjectName`	(Optional) Name of the project to which the snapshot belongs. Argument type: String
`environmentTemplateProjectName`	(Optional) Name of the project to which the environment template belongs when the snapshot is created from the environment template. Argument type: String

Positional arguments

`projectName, applicationName, snapshotName`

Response

Returns a snapshot element.

ec-perl

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

Example

```
$ec->createSnapshot ("Default", "Deploy", "Test Run 1" {description => "Test use ca
ses"});
```

ectool

syntax:]`ectool createSnapshot <projectName> <applicationName> <snapshotName> [optionals]`

Example

```
ectool createSnapshot "Default" "Deploy" "Test Run 1" --description "Test use case
s"
```

Back to Top

deleteSnapshot

Deletes snapshot from an application.

You must specify `projectName`, `applicationName`, and `snapshotName`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`applicationName`	The name of the application that must be unique among all projects. Argument type: String
`snapshotName`	The name of the snapshot that must be unique withing the application. Argument type: String

Positional arguments

`projectName, applicationName, snapshotName`

Response

None or a status OK message.

ec-perl

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

Example

```
$ec->deleteSnapshot ("Default", "Undeploy", "Pilot Version");
```

ectool

syntax: `ectool deleteSnapshot <projectName> <applicationName> <snapshotName>`

Example

```
ectool deleteSnapshot "Default" "Undeploy" "Pilot Version"
```

Back to Top

getPartialApplicationRevision

Retrieves a partial application when a snapshot is created.

You must specify `projectName` , `applicationName`, and `revisionNumber`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`applicationName`	The name of the application that must be unique among all projects. Argument type: String
`revisionNumber`	The revision number of the application. Argument type: Integer

Positional arguments

`projectName, applicationName, revisionNumber`

Response

Returns a list of environments deployed in the snapshot.

ec-perl

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

Example

```
$ec->getPartialApplicationRevision ("Pet Store", "Deploy", 2);
```

ectool

syntax: `ectool getPartialApplicationRevision <projectName> <applicationName> <revisionNumber>`

Example

```
ectool getPartialApplicationRevision "Pet Store" "Deploy" 2
```

Back to Top

getSnapshot

Find a snapshot by name.

You must specify `projectName` , `applicationName`, and `snapshotName`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`applicationName`	The application that owns the deployment history item. Argument type: String
`snapshotName`	The name of the snapshot that must be unique within the application. Argument type: String

Positional arguments

`projectName, applicationName, snapshotName`

Response

One snapshot element.

ec-perl

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

Example

```
$ec->getSnapshot ("Default", "Deploy", "Production");
```

ectool

syntax: `ectool getSnapshot <projectName> <applicationName> <snapshotName>`

Example

```
ectool getSnapshot "Default" "Deploy" "Production"
```

Back to Top

getSnapshotEnvironments

Retrieves a list of environments deployed in the specified snapshot.

You must specify `projectName` , `applicationName`, and `snapshotName`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`applicationName`	The name of the application that must be unique among all projects. Argument type: String
`snapshotName`	The name of the snapshot that must be unique within the application. Argument type: String

Positional arguments

`projectName, applicationName, snapshotName`

Response

A list of environments deployed in the snapshot.

ec-perl

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

Example

```
$ec->getSnapshotEnvironments ("Default", "Deploy", "Production");
```

ectool

syntax: `ectool getSnapshotEnvironments <projectName> <applicationName> <snapshotName>`

Example

```
ectool getSnapshotEnvironments "Default" "Deploy" "Production"
```

Back to Top

getSnapshots

Retrieves all the snapshots in an application.

You must specify `projectName` and `applicationName`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`applicationName`	The name of the application that must be unique among all projects. Argument type: String

Positional arguments

`projectName, applicationName`

Response

Zero or more snapshot elements.

ec-perl

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

Example

```
$cmdr->getSnapshots ("Default", "Deploy");
```

ectool

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

Example

```
ectool getSnapshots "Default" "Deploy"
```

Back to Top

modifySnapshot

Modifies an existing snapshot of an application.

You must specify `projectNam` , `applicationName`, and `snapshotName`.

Arguments	Descriptions
`projectName`	The name of the project that must be unique among all projects. Argument type: String
`applicationName`	The name of the application that must be unique within the project. Argument type: String
`snapshotName`	The name of the snapshot that must be unique within the application. Argument type: String

Arguments	Descriptions
`componentVersions`	(Optional) Name and version of the component for the snapshot. Use keyword `LATEST` for the latest version. Argument type: Map
`description`	(Optional) Comment text describing this object, which is not interpreted by ElectricFlow. A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`environmentName`	(Optional) Name of environment from which snapshot is created. Argument type: String
`environmentProjectName`	(Optional) Name for the project to which the environment or environment template belongs. Argument type: String
`newName`	(Optional) New name of the snapshot. Argument type: String

Positional arguments

`projectName, applicationName, snapshot`

Response

Returns a modified snapshot object.

ec-perl

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

Example

```
$ec->createSnapshot ("Default", "Deploy", "Beta Version 4", {description => "Intern
al use only"} );
```

ectool

syntax: `ectool createSnapshot <projectName> <applicationName> [optionals]`

Example

```
ectool createSnapshot "Default" "Deploy" "Beta Version 4" --description "Beta only"
```

Back to Top

API Commands - Tier Map

createTierMap

Creates a tier map for an application.

Required Arguments

`projectName`

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

Argument Type: String

`applicationName`

Description: Name of the application that must be unique among all applications in the project.

Argument Type: String

`environmentProjectName`

Description: Name of the environment's project that must be unique among all projects.

Argument Type: String

`environmentName`

Description: Name of the environment that must be unique among all applications in the project.

Argument Type: String

Optional Arguments

`applicationEntityRevisionId`

Description: Revision ID of the versioned object.

Argument Type: UUID

`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

`tierMappings`

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", "Deploy", "Default", "Prod",
{tierMapping => [{applicationTier => "Tomcat",
environmentTier => "Server 1"}, {applicationTier => "AWS",
environmentTier => "Server 2"}], tierMapName => "Web Apps"});
```

ectool

Syntax:

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

Example:

```
ectool createTierMap "Default" "Deploy" "Default" "Prod" --tierMapName "Web App
s"
--tierMapping "Tomcat"="Server 1" "AWS"="Server 2"
```

deleteTierMap

Deletes a tier map from an application.

Required Arguments

`projectName`

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

Argument Type: String

`applicationName`

Description: Name of the application that must be unique among all applications in the project.

Argument Type: String

`environmentProjectName`

Description: Name of the environment's project that must be unique among all projects.

Argument Type: String

`environmentName`

Description: Name of the environment that 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>, <environmentProjectNa
me>, <environmentName>);
```

Example:

```
$ec->deleteTierMap("Default", "Deploy", "Q4 Summary", "App Server");
```

ectool

Syntax:

```
ectool deleteTierMap <projectName> <applicationName> <environmentProjectName> <e
nvironmentName>
```

Example:

```
ectool deleteTierMap "Default" "Deploy" "Q4 Summary" "App Server"
```

deleteTierMapping

Deletes a tier mapping from a tier map.

Required Arguments

`projectName`

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

Argument Type: String

`applicationName`

Description: Name of the application that must be unique among all applications in the project.

Argument Type: String

`environmentProjectName`

Description: Name of the environment's project that must be unique among all projects.

Argument Type: String

`environmentName`

Description: Name of the environment that 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>, <environmentProje
ctName>, <environmentName>, <applicationTierName>);
```

Example:

```
$ec->deleteTierMapping("Default", "Deploy", "Q4 Summary", "PROD", "Config");
```

ectool

Syntax:

```
ectool deleteTierMapping <projectName> <applicationName> <environmentProjectNam
e> <environmentName> <applicationTierName>
```

Example:

```
ectool deleteTierMapping "Default" "Deploy" "Q4 Summary" "PROD" "Config"
```

getTierMaps

Retrieves all tier maps that are used by an application.

Required Arguments

`projectName`

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

Argument Type: String

`applicationName`

Description: Name of the application that must be unique among all projects.

Argument Type: String

Optional Arguments

`applicationEntityRevisionId`

Description: The revision ID of the versioned project.

Argument Type: UUID

`orderByEnvironmentUsage`

Description: <*Boolean flag* - `0|1|true|false`>– If this is set to `1` or `true`, the response will have the most recently used environment in the tier maps.

Argument Type: Boolean

Response

Returns a list of tier maps.

ec-perl

Syntax:

```
$<object>->getTierMaps(<projectName>, <applicationName>, {<optionals>});
```

Example:

```
$ec->getTierMaps("Default", "Take snapshot", {applicationEntityRevisionId => "4f
a765dd-73f1-11e3-b67e-b0a420524153"});
```

ectool

Syntax:

```
ectool getTierMaps <projectName> <applicationName> [optionals]
```

Example:

```
ectool getTierMaps "Default" "Take snapshot" --applicationEntityRevisionId 4fa76
5dd-73f1-11e3-b67e-b0a420524153
```

modifyTierMap

Modifies an existing tier map.

Required Arguments

`projectName`

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

Argument Type: String

`applicationName`

Description: Name of the application that must be unique among all applications in the project.

Argument Type: String

`environmentProjectName`

Description: Name of the environment's project that must be unique among all projects.

Argument Type: String

`environmentName`

Description: Name of the environment.

Argument Type: String

Optional Arguments

`applicationEntityRevisionId`

Description: Revision ID of the versioned object.

Argument Type: UUID

`tierMapName`

Description: New name of the tier map. If this argument is not specified the tier map is a hyphenated application and environment name.

Argument Type: String

`tierMappings`

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

Returns the updated tier map.

ec-perl

Syntax:

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

Example:

```
$ec->modifyTierMap("Default", "Deploy", "Utilities", "Web Server",
tierMappings => [{applicationTier => "AppTier1", environmentTier => "EnvTier1"},
{applicationTier => "AppTier2", environmentTier => "EnvTier2"}], tierMapName =>
"TierMap1"});
```

ectool

Syntax:

```
ectool modifyTierMap <projectName> <applicationName> <environmentProjectName> <e
nvironmentName> [optionals]
```

Example:

```
ectool modifyTierMap "Default" "Deploy" "Utilities" "Web Server" --tierMapName T
ierMap1 --tierMapping AppTier1=EnvTier1 AppTier2=EnvTier2
```

API Commands - User and Group Management

addUsersToGroup

Adds one or more specified users to a local group.

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

Arguments	Descriptions
`groupName`	The name of the group that must be unique among local groups.
`userName`	Using ec-perl, enter one or more user names to add to the group.
`userName`	Using ectool, enter one user name to add to the group.
`userNames`	Using ectool, enter two or more user names to add to the group.

Positional arguments

ec-perl: `groupName, userName`

ectool: `groupName, userName` or `groupName, userNames`, depending on the number of user names.

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> ...`

Examples

This example uses the singular `userName` argument to add one user to the group:

```
ectool addUsersToGroup Developers --userName John
```

This example uses the plural `userNames` argument to add more than one user to the group:

```
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`	Name for the new group that you are creating. Argument type: String
`userName`	(Optional) Using ec-perl, enter one or more user names to add to the group. Argument type: Collection
`userName`	(Optional) Using ectool, enter one user name to add to the group. Argument type: String
`userNames`	(Optional) Using ectool, enter two or more user names to add to the group. Argument type: Collection

Positional arguments

`groupName`

Response

Returns a group element.

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> [optionals]`

Examples

This example uses the singular `userName` argument to add one user name to the group.

```
ectool createGroup "Build Users" --userName "Betty Barker"
```

This example uses the plural `userNames` argument to add two or more user names to the group.

```
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
`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. Argument type: String
`email`	(Optional) The email address of the user. Argument type: String
`fullUserName`	(Optional) Full name of the user, not a nickname. Argument type: String
`groupNames`	(Optional) List of groups of which this user is a member. *<group1 group2>* Any group name containing spaces must be enclosed in double-quotes. Argument type: Collection
`password`	(Optional) The password of the user. Argument type: String

Positional arguments

`userName`

Response

Returns a `user` element.

ec-perl

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

Example

```
$cmdr->createUser("aallen", {fullUserName => "Albert Allen"});
```

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

ectool

syntax: `ectool createUser <userName> [optionals]`

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 that must be unique among local groups. Argument type: String

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. Argument type: String

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 that must be unique among local groups. Argument type: String
`providerName`	(Optional) The name of the provider from which to retrieve the group.Using this option allows you to search only the specified (LDAP or Active Directory) provider for group information. Argument type: String

Arguments	Descriptions
`traverseHierarchy`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `true` or `1`, nested groups are included in the results when the group is an LDAP group and recursive traversal of group hierarchy is enabled for the directory provider. Argument type: Boolean

Positional arguments

`groupName`

Response

One `group` element.

ec-perl

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

Example

```
$cmdr->getGroup("QA group", {providerName => "LDAP", traverseHierarchy => true});
```

ectool

syntax: `ectool getGroup <groupName> [optionals]`

Example

```
ectool getGroup "QA group" --providerName "LDAP" --traverseHierarchy true
```

Back to Top

getGroups

Retrieves all groups.

Arguments	Descriptions
`filter`	(Optional) A string used to filter the returned groups by their names. Argument type: String
`includeAll`	(Optional) <*Boolean flag* - `0|1|true|false`> When enabled (the value is `1` or `true`), this argument returns all matching groups, including LDAP or non-LDAP groups that may or may not be in the ElectricFlow database already. A group is added to the ElectricFlow database when a user (who is a member of that group) logs in to ElectricFlow for the first time. Argument type: Boolean

Arguments	Descriptions
`maximum`	(Optional) The maximum number of groups to return. Argument type: Integer

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 [optionals]`

Example

```
ectool getGroups --filter dev* --maximum 3
```

Back to Top

getUser

Retrieves information about a user.

You must specify the `userName`.

Arguments	Descriptions
`userName`	The name of the user. Argument type: String
`providerName`	(Optional) The name of the directory provider, which limits the search to the specified directory provider. Argument type: String
`traverseHierarchy`	(Optional) <*Boolean flag* - `0|1|true|false`> If this is set to `true` or `1`, parent groups are included in the result when the user is an LDAP user and recursive traversal of group hierarchy is enabled for the directory provider. Argument type: Boolean

Positional arguments

`userName`

Response

One `user` element.

ec-perl

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

Example

```
$cmdr->getUser("Betty Barker", {traverseHierarchy => true});
```

ectool

syntax: `ectool getUser <userName> [optionals]`

Example

```
ectool getUser "Betty Barker" --traverseHierarchy true
```

Back to Top

getUsers

Retrieves information about all users. By default, this command returns users who have been added to the ElectricFlow 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`	(Optional) <*filter pattern*> Enter a filter pattern to match user names. The filter is not case sensitive and can include the `"*"` wildcard character. Argument type: String
`includeAll`	(Optional) <*Boolean flag -* `0|1|true|false`> When this argument is `1` or `true`, this argument returns all matching groups, including LDAP or non-LDAP groups that may or may not be in the ElectricFlow database. A group is added to the ElectricFlow database when a user who is a member of that group logs in to ElectricFlow for the first time. Argument type: Boolean
`maximum`	(Optional) <*number of users*> Specify a larger number of user records to retrieve. The default limit is 100 user records. Argument type: Integer

Arguments	Descriptions
`searchFields`	(Optional) Filter search fields that include the user's full name and email address. Argument type: Collection

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 [optionals]`

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". Argument type: String
`userName`	The name of a user who has login privileges. Argument type: String

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`	Name of the group that must be unique among local groups. Argument type: String
`migrateSettings`	(Optional) <*targetGroupName*> Use this argument to specify the new name to which the settings need to be moved. Argument type: String
`newName`	(Optional) Enter any name of your choice to rename the group. Argument type: String
`removeAllUsers`	(Optional) <*Boolean flag* `- 0|1|true|false`> If the value is `true` or `1`, the system removes all the users from this group. Argument type: Boolean
`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. Argument type: Collection
`userName`	(Optional) Using ec-perl, enter one or more user names to add to the group. `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. Argument type: Collection
`userName`	(Optional) Using ectool, enter one user name to add to the group. `user1` Provide one user name for the group such as an alias. This name will replace existing names in the group. Any name with spaces must be enclosed in double-quotes. Argument type: String
`userNames`	(Optional) Using ectool, enter two or more user names to add to the group. `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. Argument type: Collection

Positional arguments

`groupName`

Response

An updated `group` object.

ec-perl

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

Examples

```
$cmdr->modifyGroup("Build Users", {userName => "dduncan"});
```

```
$cmdr->modifyGroup("Build Users", {userName => ["dduncan", "jack"]});
```

ectool

syntax:`ectool modifyGroup <groupName> [<optionals>]`

Examples

This example has the singular `userName` argument to add one user name to the group.

```
ectool modifyGroup "Build Users" --userName dduncan
```

This example has the plural `userNames` argument to add two or more user names to the group.

```
ectool modifyGroup "Build Users" --userNames dduncan jack tthomas
```

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
`userName`	The name used by the user to log in or receive email. For example, "jsmith". Argument type: String
`email`	(Optional) The user's email address. Argument type: String
`fullUserName`	(Optional) The user's full name. For example, "John Smith". Argument type: String
`groupNames`	(Optional) *group1* [*group2* ...] Assigns the user to one or more groups and removes the user from any groups not included in the list. Argument type: Collection
`migrateSettings`	<*targetUserName*> The new name to which the settings need to be moved. Argument type: String
`newName`	(Optional) The new name of the user (for example, changing an existing user's surname). Argument type: String **Note:** Use this option only if the existing username is not being referenced anywhere.
`password`	(Optional) Enter a new password for the user. Argument type: String
`removeFromAllGroups`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, this user will be removed from all groups. Argument type: String
`sessionPassword`	(Optional) If changing the user's password, you must also enter the password used in the "login" command. Argument type: String

Positional arguments

`userName`

Response

Returns an updated user object..

ec-perl

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

Example

```
$cmdr->modifyUser("Betty Barker", {email => "bbarker@abc.com"});
```

ectool

syntax: `ectool modifyUser <userName> [optionals]`

Example

```
ectool modifyUser "Betty Barker" --email "bbarker@abc.com"
```

Back to Top

removeUsersFromGroup

Removes one or more users from an existing local 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. Argument type: String
`userNames`	The list of users to remove from the group. Argument type: Collection

Positional arguments

`groupName, userNames`

Response

Removes one or more users from a local group.

ec-perl

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

Example

```
$cmdr->removeUsersFromGroup("Developers", {userName => ["John", "Jim", "Joey"]});
```

ectool

syntax: `ectool removeUsersFromGroup <groupName> <userNames> [optionals]`

Example

```
ectool removeUsersFromGroup Developers --userNames John Jim Joey
```

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
`projectName`	The name of the project. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`stateDefinitionName`	Choose any unique name of your choice for the state definition. This name must be unique within the workflow definition. Argument type: String

Arguments	Descriptions
`actualParameters`	(Optional) 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. Argument type: Map
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`startable`	(Optional) <*Boolean flag* - `0|1|true|false`> – A value of `1` or `true` means this state definition can be the initial state of an instantiated workflow. Argument type: Boolean
`subprocedure`	(Optional) Name of the procedure launched when the state is entered. **Also requires** `subproject`. Argument type: String
`subproject`	(Optional) Name of the project containing the procedure or workflow launched when the state is entered. Argument type: String
`substartingState`	(Optional) Name of the starting state for the workflow launched when the state is entered. **Also requires** `subproject` and `subworkflowDefinition`. Argument type: String
`subworkflowDefinition`	(Optional) Name of the workflow definition launched when the state is entered. **Also requires** `subproject`. Argument type: String

Positional arguments

`projectName`, `workflowDefinitionName`, `stateDefinitionName`

Response

One `stateDefinition` element.

ec-perl

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

Example

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

ectool

syntax: `ectool createStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> [optionals]`

Example

```
ectool createStateDefinition "Default" "BTD" "build" --startable 1 --subproject "pr
oduct"
    --subprocedure :Master" --description "Production"
```

Back to Top

createTransitionDefinition

Creates a new transition definition for workflow definition.

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

Arguments	Descriptions
`projectName`	The name for the project that must be unique among all projects. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`transitionDefinitionName`	The name of the transition that must be unique among all state definitions. Argument type: String
`targetState`	Target state for the transition definition. Argument type: String

Arguments	Descriptions
`actualParameters`	(Optional) 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. Argument type: Map
`condition`	(Optional) 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. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`trigger`	(Optional) Type of trigger for this transaction. Possible values are: `onEnter|onStart|onCompletion|manual` Argument type: TransitionTrigger

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 ("Default", "BTD", "build", "build2test", "test",
    {trigger => "manual", description => "Production"});
```

ectool

syntax:
```
ectool createTransitionDefinition <projectName> <workflowDefinitionName>
         <stateDefinitionName> <transitionDefinitionName> <targetState> [optionals]
```

Example

```
ectool createTransitionDefinition "Default" "BTD" "build" "build2test" "test" --tri
gger manual
   --description "Production"
```

Back to Top

createWorkflowDefinition

Creates a new workflow definition for a project.

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

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument type: String
`workflowDefinitionName`	Name of the workflow definition; must be unique within the project. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`workflowNameTemplate`	(Optional) The name of the workflow template that the system uses to name instances of this workflow definition. Argument type: String

Positional arguments

`projectName`, `workflowDefinitionName`

Response

One `workflowDefinition` element.

ec-perl

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

Example

```
$cmdr->createWorkflowDefinition ("Default", "BTD", {description => "Production"});
```

ectool

syntax: `ectool createWorkflowDefinition <projectName> <workflowDefinitionName> [<optionals>]`

Example

```
ectool createWorkflowDefinition "Default" "BTD" --description "Production"
```

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. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String

Positional arguments

`projectName, workflowDefinitionName, stateDefinitionName`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

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

Example

```
ectool deleteStateDefinition "Default" "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. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`transitionDefinitionName`	The name of the transition definition. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String

Positional arguments

`projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName`

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->deleteTransitionDefinition ("Default", "BTD", "build", "build2test");
```

ectool

syntax:
```
ectool deleteTransitionDefinition <projectName> <workflowDefinitionName>
        <stateDefinitionName> <transitionDefinitionName>
```

Example

```
ectool deleteTransitionDefinition "Default" "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. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String

Positional arguments

`projectName`, `workflowDefinitionName`

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->deleteWorkflowDefinition ("Default", "BTD");
```

ectool

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

Example

```
ectool deleteWorkflowDefinition "Default" "BTD"
```

Back to Top

getStateDefinition

Retrieves a state definition by its name.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`workflowDefinitionName`	Name of the workflow definition. Argument type: String
`stateDefinitionName`	Name of the state definition. Argument type: String

Positional arguments

`projectName, workflowDefinitionName, stateDefinitionName`

Response

One `stateDefinition` element.

ec-perl

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

Example

```
$cmdr->getStateDefinition ("Default", "BTD", "build");
```

ectool

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

Example

```
ectool getStateDefinition "Default" "BTD" "build"
```

Back to Top

getStateDefinitions

Retrieves all state definitions in a workflow definition.

You must specify `projectName` and `workflowDefinitionName`.

Arguments	Descriptions
`projectName`	The name for the project that must be unique among all projects. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`includeFormalParameters`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, formal parameters will also be included in the response. Argument type: Boolean
`startableOnly`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, only state definitions marked as startable will be included in the response. Argument type: Boolean

Positional arguments

```
projectName, workflowDefinitionName
```

Response

One or more `stateDefinition` elements.

ec-perl

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

Example

```
$cmdr->getStateDefinitions ("Default", "BTD", {startableOnly => 1});
```

ectool

syntax: `ectool getStateDefinitions <projectName> <workflowDefinitionName> [optionals]`

Example

```
ectool getStateDefinitions "Default" "BTD" --startableOnly 1
```

Back to Top

getTransitionDefinition

Finds a transition definition by its name.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`workflowDefinitionName`	Name of the workflow definition. Argument type: String
`stateDefinitionName`	Name of the state definition. Argument type: String
`transitionDefinitionName`	Name of the transition definition. Argument type: String

Positional arguments

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

Response

One `transitionDefinition` element.

ec-perl

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

Example

```
$cmdr->getTransitionDefinition ("Default", "BTD", "build", "build2test");
```

ectool

syntax: `ectool getTransitionDefinition <projectName> <workflowDefinitionName>`
`<stateDefinitionName> <transitionDefinitionName>`

Example

```
ectool getTransitionDefinition "Default" "BTD" "build" "build2test"
```

Back to Top

getTransitionDefinitions

Retrieves all transition definitions in a workflow definition.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`targetState`	(Optional) The name of the target state that limits results to transition definitions that have the specified state definition as a target. Argument type: String

Positional arguments

`projectName, workflowDefinitionName, stateDefinitionName`

Response

Zero or more `transitionDefinition` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getTransitionDefinitions <projectName> <workflowDefinitionName> <stateDefinitionName> [optionals]`

Example

```
ectool getTransitionDefinitions "projectA" "BTD" "build"
```

Back to Top

getWorkflowDefinition

Retrieves a workflow definition by its name.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`workflowDefinitionName`	Name of the workflow definition. Argument type: String

Positional arguments

`projectName, workflowDefinitionName`

Response

One `workflowDefinition` element.

ec-perl

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

Example

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

ectool

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

Example

```
ectool getWorkflowDefinition "Default" "BTD"
```

Back to Top

getWorkflowDefinitions

Retrieves all workflow definitions in a project.

You must specify a `projectName`.

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

Positional arguments

`projectName`

Response

Zero or more `workflowDefinition` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getWorkflowDefinitions <projectName>`

Example

```
ectool getWorkflowDefinitions "Default"
```

Back to Top

modifyStateDefinition

Modifies an existing state definition.

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

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

Arguments	Descriptions
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`stateDefinitionName`	The name of the state definition to modify. Argument type: String
`actualParameter`	(Optional) 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. Argument type: Map
`clearActualParameters`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `true` or `1`, all the actual parameters are removed from the definition. Argument type: Boolean
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`newName`	(Optional) New name for the state definition. Argument type: String
`startable`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, the workflow can begin in the specified state. Argument type: Boolean
`subprocedure`	(Optional) The name of the procedure launched when the state is entered. It also requires `subproject`. Argument type: String
`subproject`	(Optional) The name of the project containing the subprocedure or workflow that is launched when the state is entered. Argument type: String

Arguments	Descriptions
`substartingState`	(Optional) The name of the workflow starting state that is launched when the state is entered. Using this argument also requires `subproject` and `subworkflowDefinition`. Argument type: String
`subworkflowDefinition`	(Optional) The name of the workflow definition that is launched when the state is entered. It also requires `subproject`. Argument type: String

Positional arguments

`projectName, workflowDefinitionName, stateDefinitionName`

Response

One `stateDefinition` element.

ec-perl

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

Example

```
$cmdr->modifyStateDefinition ("Default", "BTD", "build",
       {startable => 1,
       subproject => "factory",
     subprocedure => "Master",
      description => "Build to deploy"});
```

ectool

syntax: `ectool modifyStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> [optionals]`

Example

```
ectool modifyStateDefinition "Default" "BTD" "build" --startable 1 --subproject fac
tory
   --subprocedure Master --description "Build to deploy"
```

Back to Top

modifyTransitionDefinition

Modifies an existing transition definition.

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

Arguments	Descriptions
`projectName`	The name for the project that must be unique among all projects. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`transitionDefinitionName`	The name of the transition definition to modify. Argument type: String
`actualParameter`	(Optional) 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`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `true` or `1`, all the actual parameters are removed from the definition. Argument type: Boolean
`condition`	(Optional) 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. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`newName`	(Optional) New name of the transition definition that must be a unique name within the workflow. Argument type: String
`targetState`	(Optional) The target state for the transition definition. Argument type: String

Arguments	Descriptions
`trigger`	(Optional) Possible values are: `onEnter|onStart|onCompletion|manual`. Argument type: TransitionTrigger

Positional arguments

`projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName`

Response

One `transitionDefinition` element.

ec-perl

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

Example

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

ectool

syntax: `ectool modifyTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName> [optionals]`

Example

```
ectool modifyTransitionDefinition "Default" "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
`projectName`	Name for the project that must be unique among all projects. Argument type: String

Arguments	Descriptions
`workflowDefinitionName`	The name of the workflow definition to modify. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`newName`	(Optional) The new name for the workflow definition. It must be a unique name within the workflow. Argument type: String
`workflowNameTemplate`	(Optional) The template used to determine default names for workflows launched from a workflow definition. Argument type: String

Positional arguments

`projectName, workflowDefinitionName`

Response

One `workflowDefinition` element.

ec-perl

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

Example

```
$cmdr->modifyWorkflowDefinition ("Default", "BTD",
        {newName => "Build Test Deploy",
     description => "changed name"});
```

ectool

syntax: `ectool modifyWorkflowDefinition <projectName> <workflowDefinitionName> [optionals]`

Example

```
ectool modifyWorkflowDefinition "Default" "BTD"
   --newName "Build Test Deploy"
   --description "changed name"
```

Back to Top

moveStateDefinition

Moves a state definition within a workflow definition.

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

Arguments	Descriptions
`projectName`	The name of the project containing the state definition. Argument type: String
`stateDefinitionName`	The name of the state definition to move. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`beforeStateDefinition`	(Optional) 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. Argument type: String

Positional arguments

`projectName, workflowDefinitionName, stateDefinitionName`

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->moveStateDefinition ("Default", "BTD", "Unit test",
     {beforeStateDefinition => "Build"});
```

ectool

syntax: `ectool moveStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> [optionals]`

Example

```
ectool moveStateDefinition "Default" "BTD" "Unit test" --beforeStateDefinition "Bui
ld"
```

Back to Top

moveTransitionDefinition

Moves a transition definition within a workflow definition.

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

Arguments	Descriptions
`projectName`	The name of the project containing the transition definition. Argument type: String
`workflowDefinitionName`	The name of the workflow definition. Argument type: String
`stateDefinitionName`	The name of the state definition. Argument type: String
`transitionDefinitionName`	The name of the transition definition to move. Argument type: String
`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.

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 ("Default", "BTD", "Build", "in",
    {beforeTransitionDefinition => "out"});
```

ectool

syntax: `ectool moveTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName> [optionals]`

Example

```
ectool moveTransitionDefinition "Default" "BTD" "Build" "in" --beforeTransitionDefi
nition "out"
```

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 for the project that must be unique among all projects. Argument type: String
`workflowName`	The name of the workflow. Argument type: String

Positional arguments

```
projectName, workflowName
```

Response

None or status OK message.

ec-perl

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

Example

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

ectool

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

Example

```
ectool completeWorkflow Default 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
`projectName`	The name of the project containing the workflow to delete. Argument type: String
`workflowName`	The name of the workflow. Argument type: String
`deleteProcesses`	(Optional) <*Boolean flag -* `0|1|true|false`> If the value is `1` or `true`, the processes associated with the workflow will also be deleted. Argument type: Boolean

Positional arguments

`projectName, workflowName`

Response

None or status OK message.

ec-perl

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

Example

```
$cmdr->deleteWorkflow ("Default", "workflow_26_201010121647", {deleteProcesses => t
rue});
```

ectool

syntax: `ectool deleteWorkflow <projectName> <workflowName> [optionals]`

Example

```
ectool deleteWorkflow "Default" "workflow_26_201010121647" --deleteProcesses true
```

Back to Top

getState

Finds a state by its name.

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

Arguments	Descriptions
`projectName`	Name for the project; must be unique among all projects. Argument type: String
`workflowName`	Name of the workflow. Argument type: String
`stateName`	Name of the state. Argument type: String

Positional arguments

`projectName, workflowName, stateName`

Response

One `state` element.

ec-perl

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

Example

```
$cmdr->getState ("Default", "workflow_26_201010121647", "build");
```

ectool

syntax: `ectool getState <projectName> <workflowName> <stateName>`

Example

```
ectool getState "Default" "workflow_26_201010121647" "build"
```

Back to Top

getStates

Retrieves all states in a workflow.

You must specify `projectName` and `workflowName`.

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`workflowName`	Name of the workflow. Argument type: String

Positional arguments

`projectName, workflowName`

Response

One or more `state` elements.

ec-perl

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

Example

```
$cmdr->getStates ("Default", "workflow_26_201010121647");
```

ectool

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

Example

```
ectool getStates "Default" "workflow_26_201010121647"
```

Back to Top

getTransition

Finds a transition by its name.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`workflowName`	Name of the workflow. Argument type: String
`stateName`	Name of the new state. Argument type: String

Arguments	Descriptions
`transitionName`	Name of the transition instance. Argument type: String

Positional arguments

`projectName, workflowName, stateName, transitionName`

Response

One `transition` element.

ec-perl

syntax: `$cmdr->getTransition (projectName>, <workflowName>, <stateName>, <transitionName>);`

Example

```
$cmdr->getTransition ("Default", "workflow_26_201010121647", "build", "build2tes
t");
```

ectool

syntax: `ectool getTransition <projectName> <workflowName> <stateName> <transitionName>`

Example

```
ectool getTransition "Default" "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 that must be unique among all projects. Argument type: String
`workflowName`	The name of the workflow. Argument type: String
`stateName`	The name of the new state. Argument type: String

Arguments	Descriptions
`targetState`	(Optional) The name of the target state that limits results to transitions that have the specified state as a target. Argument type: String

Positional arguments

`projectName, workflowName, stateName`

Response

One or more `transition` elements.

ec-perl

syntax: `$cmdr->getTransitions (<projectName>, <workflowName>, <stateName>, {<optionals>});`

Example

```
$cmdr->getTransitions ("Default", "workflow_26_201010121647", "build");
```

ectool

syntax: `ectool getTransitions <projectName> <workflowName> <stateName> [optionals]`

Example

```
ectool getTransitions "Default" "workflow_26_201010121647" "build"
```

Back to Top

getWorkflow

Finds a workflow by its name.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`workflowName`	Name of the workflow. Argument type: String

Positional arguments

`projectName, workflowName`

Response

One `workflow` element.

ec-perl

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

Example

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

ectool

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

Example

```
ectool getWorkflow "Default" "BTD"
```

Back to Top

getWorkflows

Retrieves all workflow instances in a project.

You must specify a `projectName`.

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

Positional arguments

`projectName`

Response

Zero or more `workflow` elements.

ec-perl

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

Example

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

ectool

syntax: `ectool getWorkflows <projectName>`

Example

```
ectool getWorkflows "Default"
```

Back to Top

runWorkflow

Runs the specified workflow definition and returns the workflow name.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`workflowDefinitionName`	Name of the workflow definition. Argument type: String
`actualParameter`	(Optional) 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. Argument type: Map
`credentials`	(Optional) Credentials to use with the workflow state. Argument type: Collection
`priority`	(Optional) Priority of the jobs launched by the workflow. Argument type: JobPriority
`scheduleName`	(Optional) Name for the schedule for environment reservations that must be unique among all schedules for the project. Argument Type: String
`startingState`	(Optional) The initial state of the workflow. Argument type: String

Positional arguments

`projectName, workflowDefinitionName`

Response

Returns the workflow name.

ec-perl

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

Example

```
$cmdr->runWorkflow ("Default", "BTD", {startingState => "Build"});
```

ectool

syntax: `ectool runWorkflow <projectName> <workflowDefinitionName> [optionals]`

Example

```
ectool runWorkflow "Default" "BTD" --startingState "Build"
```

Back to Top

transitionWorkflow

Manually transition from the active workflow state.

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

Arguments	Descriptions
`projectName`	Name for the project that must be unique among all projects. Argument type: String
`workflowName`	The name of the workflow to transition. Argument type: String
`stateName`	The name of the new state. Argument type: String
`transitionName`	The name of the transition. Argument type: String
`actualParameters`	(Optional) 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. Argument type: Map
`credentials`	(Optional) Credentials to use with the workflow state. Argument type: Collection

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 ("Default", "workflow_26_201010121647", "build", "build2t
est");
```

ectool

syntax: `ectool transitionWorkflow <projectName> <workflowName> <stateName> <transitionName> [optionals]`

Example

```
ectool transitionWorkflow "Default" "workflow_26_201010121647" "build" "build2test"
```

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.

agentDrivePath	agentUncPath	Result from running a step in "job123" that uses this workspace
`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
`workspaceName`	The workspace name. Argument type: String
`agentDrivePath`	(Optional) Drive-letter-based path used by Windows agents to access the workspace in steps. Argument type: String
`agentUncPath`	(Optional) UNC path used by Windows ElectricFlow 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. Argument type: String
`agentUnixPath`	(Optional) UNIX path used by UNIX agents and Linux ElectricFlow web servers to access the workspace. Argument type: String
`credentialName`	(Optional) 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. Argument type: String

Arguments	Descriptions
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`local`	(Optional) <*Boolean flag* - `0|1|true|false`> When the value is `1` or `true`, the workspace is local. Argument type: String
`workspaceDisabled`	(Optional) <*Boolean flag* - `0|1|true|false`> When the value is `1` or `true`, the workspace is disabled. Argument type: String
`zoneName`	(Optional) The name of the zone where this workspace resides. Argument type: String

Positional arguments

`workspaceName`

Response

Returns a workspace element.

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> [optionals]`

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. Argument type: String

Positional arguments

`workspaceName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteWorkspace("Test summaries");
```

ectool

syntax: `ectool deleteWorkspace <workspaceName>`

Example

```
ectool deleteWorkspace "Test summaries"
```

Back to Top

getWorkspace

Retrieves a workspace by its name.

You must specify the `workspaceName`.

Arguments	Descriptions
`workspaceName`	The name of the workspace. Argument type: String

Positional arguments

`workspaceName`

Response

One `workspace` element.

ec-perl

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

Example

```
$cmdr->getWorkspace("Test results");
```

ectool

syntax: `ectool getWorkspace <workspaceName>`

Example

```
ectool getWorkspace "Test results"
```

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
`workspaceName`	The name of the workspace to modify. Argument type: String
`agentDrivePath`	(Optional) Drive-letter-based path used by Windows agents to access the workspace in steps. Argument type: String
`agentUncPath`	(Optional) UNC path used by Windows ElectricFlow 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. Argument type: String
`agentUnixPath`	(Optional) UNIX path used by UNIX agents and Linux ElectricFlow web servers to access the workspace. Argument type: String

Arguments	Descriptions
`credentialName`	(Optional) `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. Argument type: String
`description`	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>` Argument type: String
`local`	(Optional) <*Boolean flag* - `0|1|true|false`> If the value is `1` or `true`, the workspace is local. Argument type: Boolean
`newName`	(Optional) Enter any name of your choice to rename the workspace. Argument type: String
`workspaceDisabled`	(Optional) <*Boolean flag* - `0|1|true|false`> If the value is `1` or `true`, the workspace is disabled. Argument type: Boolean
`zoneName`	(Optional) The name of the zone where this workspace resides. Argument type: String

Positional arguments

`workspaceName`

Response

Returns a `workspace` element.

ec-perl

syntax: `$cmdr->modifyWorkspace(<workspaceName>, {<optionals>});`

Example

```
$cmdr->modifyWorkspace("Test", {description => "QA test workspace"});
```

ectool

syntax: `ectool modifyWorkspace <workspaceName> [optionals]`

Example

```
ectool modifyWorkspace "Test" --description "QA test workspace"
```

Back to Top

resolveFile

Resolves the path to a log file or artifact in a workspace.

You must specify `fromAgentId` and `workspaceName`.

Arguments	Descriptions
`fromAgentId`	Identifier of the agent requesting the route to a destination agent or artifact repository. Argument type: Long
`workspaceName`	Name of the workspace. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) 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. Argument type: UUID
`resourceName`	(Optional) Name of the resource. Argument type: String

Positional arguments

`fromAgentId, workspaceName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->resolveFile ( <fromAgentId>, <workspaceName>, {<optionals>});`

Example

```
$cmdr->resolveFile ("Machine 2", "Dev shared workspace", {resourceName=> "Server 1"
});
```

ectool

syntax: `ectool resolveFile <fromAgentId> <workspaceName> [optionals]`

Example

```
ectool resolveFile "Machine 2" "Dev shared workspace" --resourceName "Server 1"
```

Back to Top

API Commands - Miscellaneous Management

acquireNamedLock

Retrieves the named lock.

Arguments	Descriptions
`lockName`	Name of the lock. Argument type: String
`create`	<*Boolean flag* - `0|1|true|false`> When this argument is set to `true` or `1`, the system will create a lock if it does not exist. Argument type: Boolean

Positional arguments

`lockName, create`

Response

None or a status OK message.

ec-perl

syntax:`$cmdr->acquireNamedLock(<lockName>, <create>);`

Example

```
$cmdr->acquireNamedLock ("Group2", true);
```

ectool

syntax:`ectool acquireNamedLock <lockName> <create>`

Example

```
ectool acquireNamedLock "Group 2" true
```

Back to Top

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 the owner of an object. For email notifiers, the owner can be changed if the current user has sufficient privileges to delete and recreate the object.

Arguments	Descriptions
`applicationName`	(Optional) The name of the application. Argument type: String

Arguments	Descriptions
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration. Argument type: String
`credentialName`	(Optional) `credentialName` can be in one of these formats: • **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. Argument type: String
`environmentName`	(Optional) The name of an environment. Argument type: String
`environmentTemplateName`	(Optional) The name of an environment template. Argument type: String
`environmentTierName`	(Optional) The name of an environment tier. Argument type: String
`groupName`	(Optional) The full name of a group. For Active Directory and LDAP, this is a full domain name. Argument type: String
`newOwnerName`	(Optional) The name of the new owner for this object. This defaults to the current user. Argument type: String
`notifierName`	(Optional) The name of the email notifier. Argument type: String

Arguments	Descriptions
`pluginName`	(Optional) The name of the plugin. This is the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin. Argument type: String
`procedureName`	(Optional) The name of the procedure. It can be a path to the procedure. When using this argument, you must also enter the `projectName`. Argument type: String
`processName`	(Optional) The name of a process. It can be a path to the process. Argument type: String
`processStepName`	(Optional) The name of a process step. It can be a path to the process step. Argument type: String
`projectName`	(Optional) The name of the project. It can be a path to the project. The project name is ignored for credentials, procedure, steps, and schedules when it is specified as a path. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`resourceName`	(Optional) The name of the resource. Argument type: String
`scheduleName`	(Optional) The name of the schedule. It can be a path to the schedule. When using this argument, you must also use `projectName`. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stepName`	(Optional) The name of the step. It can be a path to the step. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String

Arguments	Descriptions
`userName`	(Optional) The full name of the user. For Active Directory and LDAP, the name can be `user@domain`. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workspaceName`	(Optional) The name of the workspace. Argument type: String

Positional arguments

None

Response

Returns the modified object.

ec-perl

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

Example

```
$cmdr->changeOwner ({"projectName" => "Default"});
```

ectool

syntax: `ectool changeOwner [optionals]`

Example

```
ectool changeOwner --projectName "Default"
```

Back to Top

clone

Makes a copy of an existing ElectricFlow object. These objects can be cloned:

- Applications
- Components
- Credentials
- Directory providers
- Email configurations
- Email notifiers
- Environment templates

- Flow
- Flow states
- Flow transitions
- Pipelines
- Procedures
- Procedure steps
- Processes
- Projects
- Property sheets
- Releases
- Resources
- Resource pools
- Resource templates
- Schedules
- Stages
- State definitions
- Transition definitions
- Workflow definitions
- Workspaces

IMPORTANT: You cannot clone parameters, artifacts, or artifact versions.

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 the objects that can be cloned

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`	(Optional) New name of the cloned copy of an object. 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. Argument type: String

Arguments	Descriptions
`Optionals`	
`disableProjectTracking`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `true`, when copying a project, even if Change Tracking is enabled for the original project, Change Tracking is disabled for the new copy of the project from its creation. If you do not need to track changes for the new copy, this avoids the Change Tracking overhead that would otherwise slow down the copying operation, and also saves having to subsequently disable Change Tracking for the new copy of the project. Argument type: Boolean
`reducedDetailChangeHistory`	(Optional) <*Boolean flag* - `0|1|true|false`> Use this argument for large projects containing over 20,000 audited objects with Change Tracking enabled. When this argument is set to `true` or `1`, ElectricFlow automatically decreases the amount of Change History indexing information that it saves in a large project, reducing the level of detail for Change Tracking-intensive operations in the Change History. This can make it harder to revert an object to a specific state and to find information in the Change History when you are troubleshooting or debugging an issue. Set this argument to `false` or `0` to suppress to this behavior so that ElectricFlow does not change the amount of indexing information for a large project. This will cause the operation to take longer and put more load on the database, but the Change History will have the full details of the entities owned by objects in the project. Argument type: Boolean
`Locators`	
`applicationName`	(Optional) The name of the application that is unique among all projects. Argument type: String
`componentName`	(Optional) The name of the component. Argument type: String
`configName`	(Optional) The name of the email configuration (`emailConfig`). Argument type: String

Arguments	Descriptions
`credentialName`	(Optional) The name of the credential that can be specified in one of these formats: **relative** (for example, *"cred1"*)–The credential is assumed to be in the project that contains the target object. **absolute**(for example, *"/projects/BuildProject/credentials/cred1"*)–Tthe credential can be from any specified project, regardless of the project with the target object. Argument type: String
`environmentTemplateName`	(Optional) The name of the environment template. Argument type: String
`flowName`	(Optional) Name of the flow that must be unique within the project. Argument Type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) Name of the flow state that must be unique within the flow. Argument Type: String
`flowTransitionName`	Name of the flow transition that must be unique within the flow state. Argument Type: String
`notifierName`	(Optional) The name of the email notifier (`emailNotifier`). Argument type: String
`objectId`	(Optional) The object id as returned by `findObjects`. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin. Argument type: String

Arguments	Descriptions
`procedureName`	(Optional) The name of the procedure that you want to clone. When using this argument, you must also enter the `projectName`. Argument type: String
`processName`	(Optional) The name of the process. Argument type: String
`projectName`	(Optional) The name of the project that you want to clone. **IMPORTANT:** When an application, pipeline, or release is cloned across different projects, you may have to fix the references after the objects are cloned. For example, if a pipeline with an application reference in "default" project is cloned to a different project, the application reference needs to be fixed after the pipeline is cloned. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created.
`providerName`	(Optional) The unique name of the directory provider, such as the LDAP or Active Directory provider name. Argument type: String
`releaseName`	(Optional) The name of the Release which owns the property. Argument type: String
`resourceName`	(Optional) The name of the resource that you want to clone. Argument type: String
`resourcePoolName`	(Optional) The name of a resource pool. Argument type: String
`resourceTemplateName`	(Optional) The name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of the schedule that you want to clone. When using this argument, you must also enter `projectName`. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String

Arguments	Descriptions
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stepName`	(Optional) The name of the procedure step that you want to clone. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String
`transitionDefinitionName`	(Optional) The name of the transition definition. Argument type: String
`workflowDefinitionName`	(Optional) The name of the workflow definition. Argument type: String
`workspaceName`	(Optional) The name of the workspace that you want to clone. Argument type: String

Positional arguments

None.

To clone an existing entity, you must specify the following arguments:

- A new name for the cloned object (`cloneName`)
- Locator arguments for the objects that can be cloned

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 ({<optionals>});`

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 [optionals]`

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

Returns the count of objects specified by the provided filter.

You must enter `objectType`.

Arguments	Descriptions
`objectType`	The type of object to count. Values include: application applicationTier artifact artifactVersion component credential deployerApplication deployerConfiguration directoryProvider emailConfig emailNotifier environment environmentTemplate environmentTemplateTier environmentTier formalParameter job jobStep logEntry pipeline plugin procedure procedureStep process processDependency processStep project property release repository resource resourcePool resourceTemplate schedule snapshot stage state stateDefinition task tierMap transition transitionDefinition workflow workflowDefinition workspace Argument type: String

Arguments	Descriptions

filter	(Optional) 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 ElectricFlow 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 ElectricFlow 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)`
	
	Argument type: Collection

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

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
`objectType`	This argument specifies the type of object to find. Values include: `artifact|artifactVersion|job|logEntry|project|repository|workflow` Argument type: String

Arguments	Descriptions

`filters`	(Optional) 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 ElectricFlow 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 ElectricFlow 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)`
	
	Argument type: Collection

Arguments	Descriptions
`maxIds`	(Optional) `<id count>` The maximum number of objects that will be deleted. The default is all objects that match the filter. Argument type: Integer
`sorts`	(Optional) 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 about forming the list and passing it to the ElectricFlow Perl API. The sort order affects which objects are deleted if a `maxIds` value limits the number of objects returned by the filter. Argument type: Collection

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

dumpHeap

Captures a Java heap dump.

Arguments	Descriptions
`fileName`	Name of the file to which the heap dump is written. Argument type: String

Positional arguments

`fileName`

Response

None or a status OK message.

ec-perl

syntax:`$cmdr->dumpHeap (<fileName>);`

Example

```
$cmdr->dumpHeap ('$[/myWorkspace/data]/$[JavaFileName]');
```

ectool

syntax:`ectool dumpHeap <fileName>`

Example

```
ectool dumpHeap '$[/myWorkspace/data]/$[JavaFileName]'
```

Back to Top

dumpStatistics

Prints (emits) internal timing statistics.

Arguments	Descriptions
`clearStatistics`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, the system clears the statistics after logging them. Argument type: Boolean
`dumpLapTimes`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, the system dumps lap times. If this argument is set to `0` or `false`, the system dumps global times. Argument type: Boolean
`fileName`	(Optional) If you specify a file name with a path, the output is redirected to this file. When the path is relative, the file is written relative to the working directory of the server. Argument type: String
`format`	(Optional) Format of the output. Valid values are `text` or `xml`. The default is . Argument type: String `text`

Positional arguments

None

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->dumpStatistics ({clearStatistics=>true, format=>xml});
```

ectool

syntax:`ectool dumpStatistics [optionals]`

Example

```
ectool dumpStatistics --clearStatistics true --format xml
```

Back to Top

evalDsl

Evaluates and runs an ElectricFlow domain-specific language (DSL) script.

You must enter either the `dsl` or the `dslFile` argument.

Arguments	Descriptions
`dsl`	The DSL text. Either `dsl` or `dslFile` must be specified. Argument type: String
`dslFile`	Path to the file on the client containing the DSL script. Either `dsl` or `dslFile` must be specified. Argument type: String
`applicationName`	(Optional) Name of the application. Argument type: String
`applicationTierName`	(Optional) Name of the application tier. Argument type: String
`artifactName`	(Optional) Name of the artifact. Argument type: String
`artifactVersionName`	(Optional) Name of the artifact version. Argument type: String
`componentName`	(Optional) Name of the component. Argument type: String
`configName`	(Optional) Name of the email configuration. Argument type: String
`credentialName`	(Optional) Name of the credential. Argument type: String
`debug`	(Optional) *<Boolean flag -* `0|1|true|false`*>* If this argument is set to `true` or `1`, ElectricFlow generates debug output as the DSL script is evaluated. Argument type: Boolean

Arguments	Descriptions
`describe`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `true` or `1`,ElectricFlow prints a description of the DSL text. Argument type: Boolean
`environmentName`	(Optional) Name of the environment. Argument type: String
`environmentTemplateName`	(Optional) Name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) Name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) Name of the environment tier. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of the group. Argument type: String
`jobId`	(Optional) The primary key or name of the job. Argument type: UUID

Arguments	Descriptions
`jobStepId`	(Optional) The primary key or name of the job step. Argument type: UUID
`notifierName`	(Optional) The name of the notifier. Argument type: UUID
`objectId`	(Optional) The object ID returned by the `findObjects` call. Argument type: UUID
`parameters`	(Optional) Parameters are passed to the script by ElectricFlow as JSON text. Argument type: String
`parametersFile`	(Optional) Path to the file on the client containing parameters as JSON text that should be passed to the DSL script by ElectricFlow. Argument type: String
`path`	(Optional) The property path. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of the plugin. Argument type: String
`procedureName`	(Optional) The name of the procedure. Argument type: String
`processName`	(Optional) The name of the process if the container is a process or process step. Argument type: String
`processStepName`	(Optional) The name of the process step if the container is a process step. Argument type: String
`projectName`	(Optional) The name of the project. Argument type: String
`propertySheetId`	(Optional) The primary key or name of the property sheet. Argument type: UUID

Arguments	Descriptions
`releaseName`	(Optional) The name of the Release. Argument type: String
`repositoryName`	(Optional) The name of the repository. Argument type: String
`resourceName`	(Optional) The name of the resource. Argument type: String
`resourcePoolName`	(Optional) The name of the resource pool. Argument type: String
`resourceTemplateName`	(Optional) The name of the resource template. Argument type: String
`scheduleName`	(Optional) The name of the schedule. Argument type: String
`serverLibraryPath`	(Optional) Path to the server directory that contains `.jar` files and classes to be added to the classpath when evaluating the DSL text. Argument type: String
`sessionPassword`	(Optional)The session user's password that is used to reverify the user's identity before changing user passwords in the system. Argument type: String
`snapshotName`	(Optional) The name of the snapshot that you want to clone. Argument type: String
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step that you want to clone. When using this argument, you must also enter `projectName` and `procedureName`. Argument type: String

Arguments	Descriptions
systemObjectName	(Optional) System object names include: `admin|artifacts|directory|emailConfigs|forceAbort|licensing|log|plugins|priority|projects|repositories|resources|server|session|workspaces|zonesAndGateways` Argument type: SystemObjectName
taskName	(Optional) The name of the task. Argument type: String
timeout	(Optional) The maximum duration of execution for the specified DSL, in seconds. This defaults to `maxDslDuration` server setting. The default `timeout` is 5 minutes. Argument type: Long
transitionDefinitionName	(Optional) The name of the transition definition. Argument type: String
transitionName	(Optional) The name of the transition. Argument type: String
userName	(Optional) The name of the user where you may need to expand the string. Argument type: String
workflowDefinitionName	(Optional) The name of the workflow definition. Argument type: String
workflowName	(Optional) The name of the workflow. Argument type: String
workspaceName	(Optional) The name of the workspace that you want to clone. Argument type: String
zoneName	(Optional) The name of the zone. Argument type: String

Positional arguments

`dsl`

Response

None or a status OK message.

ec-perl

syntax:`$cmdr->evalDsl(<dsl or dslFile>, {<optionals>});`

Example

When specifying the DSL text (`<dsl>`):

```
$cmdr->evalDsl "project 'My WAR file' "
```

When specifying the path to the DSL script on the client (`<dslFile>`:

```
$cmdr->evalDsl ({dslFile => c:/dslScripts/myWarFile.dsl});
```

ectool

syntax:`ectool evalDsl <dsl> [optionals]`

Example

When specifying the DSL text (`<dsl>`):

```
ectool evalDsl "project 'My WAR file' "
```

When specifying the path to the DSL script on the client (`<dslFile>`:

```
ectool evalDsl --dslFile c:/dslScripts/myWarFile.dsl
```

Back to Top

evalScript

Evaluates a script in the specified 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`.

Arguments	Descriptions
`value`	The script to evaluate in the specified context. Argument type: String
`applicationName`	(Optional) The name of the application that must be unique among all projects. Argument type: String
`applicationTierName`	(Optional) The name of the application tier. Argument type: String
`artifactName`	(Optional) The name of the artifact. Argument type: String

Arguments	Descriptions
`artifactVersionName`	(Optional) 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 ElectricFlow server interprets either name form correctly. Argument type: String
`componentName`	(Optional) Name of the component. Argument type: String
`configName`	(Optional) Name of the email configuration. Argument type: String
`credentialName`	(Optional) Name of the credential in one of these 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"*)–Tthe credential can be from any specified project, regardless of the target object's project. Argument type: String
`environmentName`	(Optional) The name of the environment that must be unique among all projects. Argument type: String
`environmentTemplateName`	(Optional) The name of the environment template. Argument type: String
`environmentTemplateTierName`	(Optional) The name of the environment template tier. Argument type: String
`environmentTierName`	(Optional) The name of the environment tier. Argument type: String
`flowName`	(Optional) The name of the flow. Argument type: String
`flowRuntimeName`	(Optional) Name of the flow runtime. Argument Type: String

Arguments	Descriptions
`flowRuntimeStateName`	(Optional) Name of the flow state. Argument Type: String
`flowStateName`	(Optional) The name of the flow state. Argument type: String
`flowTransitionName`	(Optional) The name of the flow transition. Argument type: String
`gatewayName`	(Optional) The name of the gateway. Argument type: String
`groupName`	(Optional) The name of a group where you might evaluate a script. Argument type: String
`jobId`	(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`jobStepId`	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
`notifierName`	(Optional) The name of the email notifier. Argument type: String
`objectId`	(Optional) This is an object identifier returned by `findObjects` and `getObjects`. Argument type: String
`path`	(Optional) The property path. Argument type: String
`pipelineName`	(Optional) The name of the pipeline. Argument type: String
`pluginName`	(Optional) The name of a plugin where you might evaluate a script. Argument type: String

Arguments	Descriptions
`procedureName`	(Optional) The name of a procedure where you might need to evaluate a script. **Also requires** `projectName`. Argument type: String
`processName`	(Optional) The name of the process when the container is a process or process step. Argument type: String
`processStepName`	(Optional) The name of the process step when the container is a process step. Argument type: String
`projectName`	(Optional) Name of the project that contains the script to evaluate. Argument type: String
`propertySheetId`	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
`releaseName`	(Optional) The name of the Release. Argument type: String
`repositoryName`	(Optional) The name of the repository for artifact management. Argument type: String
`resourceName`	(Optional) The name of a resource where you might evaluate a script. Argument type: String
`resourcePoolName`	(Optional) The name of a pool containing one or more resources. Argument type: String
`resourceTemplateName`	(Optional) The name of the resource template. Argument type: String
`scheduleName`	(Optional)The name of a schedule within this project. **Also requires** `projectName`. Argument type: String
`snapshotName`	(Optional) The name of the snapshot. Argument type: String

Arguments	Descriptions
`stageName`	(Optional) The name of the stage definition. Argument type: String
`stateDefinitionName`	(Optional) The name of the state definition. Argument type: String
`stateName`	(Optional) The name of the state. Argument type: String
`stepName`	(Optional) The name of the step whose script you might evaluate. **Also requires** `projectName` and `procedureName`. Argument type: String
`systemObjectName`	(Optional) System object names include: `admin|directory|log|priority|projects|resources|server|session|workspaces`. Argument type: SystemObjectName
`taskName`	(Optional) The name of the task. Argument type: String
`transitionDefinitionName`	(Optional) Name of the transition definition. Argument type: String
`transitionName`	(Optional) Name of the transition. Argument type: String
`userName`	(Optional) Name of the user where you may need to evaluate a script. Argument type: String
`workflowDefinitionName`	(Optional) Name of the workflow definition. Argument type: String
`workflowName`	(Optional) Name of the workflow. Argument type: String
`workspaceName`	(Optional) Name of a workspace where you may need to evaluate a script. Argument type: String
`zoneName`	(Optional) Name of the zone. Argument type: String

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>, {<optionals>});`

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> [optionals]`

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

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.

The export operation is run by the server process, not through ectool. The command is not run by the agent, but by the server itself. Therefore, it has some impact if the server agent service user and the server service user are different. For example, the following commands in the same step are executed by two different users:

```
mkdir ("/path/foo");
$ec->export("/path/foo/project.xml",
{path=>"/projects/MYPROJ"}
);
```

The /path/foo directory creation is executed by the agent service, which means that the agent user needs permission to create the directory. The export is executed by the ElectricFlow service user.

Note: Before you perform an export, ensure that the ElectricFlow server is inactive (meaning that it cannot accept jobs) by completing the following steps on the server:

1. Disable ElectricSentries.
2. Disable project schedules.
3. Check that no jobs are running on any resources.
4. Disable all resources so that no new job steps can run.

This ensures a complete XML file by preventing changes to the Flow DB during the export.

IMPORTANT: When you specify the path, this is the path relative to the server process on the server host. The export operation is run using the server's process ID that must have write permission to this path.

If a relative file name is specified, the file is created relative to the ElectricFlow 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`.

The default timeout is 10800 seconds (180 minutes or 3 hours).

Note: A full export/import preserves job IDs, but a partial import preserves names only, not IDs.

IMPORTANT: When you export a project while a pipeline is in progress, only the full export includes flow runtimes that have been completed. If you want to include in-progress pipeline runs in the path-to-production view and the visual indicators showing their percentage completed in the Release dashboard, set the `excludeJobs` argument to `0` or `false` in the `export` command. When the XML file is imported, the in-progress pipeline runs in the imported project are displayed in the path-to-production view and the Release dashboard. The jobs might be incomplete once the XML is imported.

Arguments	Descriptions
`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. Argument type: String
`compress`	(Optional) <*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 file name. The default behavior is to compress the output. **Note:** This is true only for full exports. Argument type: Boolean

Arguments	Descriptions
`disableProjectTracking`	(Optional) <*Boolean flag -* `0|1|true|false`> If set to `1` or `true` when importing or exporting a project, even if the original project had Change Tracking enabled, make Change Tracking of the newly imported or exported project be disabled from its creation. If you do not need to track changes to the new project, this avoids the Change Tracking overhead that would otherwise slow down the import operation, and also saves having to subsequently disable change tracking of the re-imported project. **Note:** This is true only for partial exports. Argument type: Boolean
`download`	(Optional) <*Boolean flag -* `0|1|true|false`> If set to `1` or `true`, the exported file can be downloaded. **Note:** This is true for full and partial exports. Argument type: Boolean
`excludeJobs`	(Optional) <*Boolean flag -* `0|1|true|false`> If set to `1` or `true`, no job information will be exported. This argument can be used to reduce the size of the export file. **Note:** This is true only for full exports. Argument type: Boolean
`objectId`	(Optional) ID of the object ID. **Note:** This triggers a partial export. Argument type: UUID
`objectType`	(Optional) The object type. **Note:** This triggers a partial export. Argument type: String
`path`	(Optional) <*property path*> Specifies the path relative to the server process 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. **Note:** This triggers a partial export. Argument type: String

Arguments	Descriptions
`reducedDetailChangeHistory`	(Optional) <*Boolean flag* - `0|1|true|false`> Setting this argument to `0` or `false`prevents importing a large project that has change tracking enabled from automatically reducing the level of detail that it tracks for the change history of the import. This will make the import operation take longer and put more load on the database, but will allow the Change History feature to still show the full detail of entities owned by the imported project that were created by the import operation. Argument type: Boolean
`relocatable`	(Optional) <*Boolean flag* - `0|1|true|false`> If the `--relocatable` flag is set to `1` or `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. Argument type: Boolean
`revisionNumber`	(Optional) Revision number of the file. **Note:** This triggers a partial export. Argument type: Integer
`safeMode`	(Optional) 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 affects only full imports. Argument type: SafeMode
`withAcls`	(Optional) This argument modifies `relocatable`. <*Boolean flag* - `0|1|true|false`> If the `withAcls` flag is set to `1` or `true`, a relocatable partial export will include ACLs. **Note:** This is true only for partial exports. Argument type: Boolean

Arguments	Descriptions
`withNotifiers`	(Optional) This argument modifies `relocatable`. *<Boolean flag* - `0|1|true|false`> If this flag is set to `1` or `true`, a relocatable partial export will include email notifiers. **Note:** This is true only for partial exports. Argument type: Boolean
`withVersionNumbers`	(Optional) *<Boolean flag* - `0|1|true|false`> If this flag is setto `1` or `true`, version numbers are issued during the export operation (they are ignored on import). **Note:** This is true only for partial exports. Argument type: Boolean

Positional arguments

`fileName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->export(<fileName>, {<optionals>});`

Examples

```
$cmdr->export("c:\ElectricCommander\Aug 15 2015.xml");

$cmdr->export("c:\ElectricCommanderBackup\Test Proj.xml",
                    {path => "/projects[Test Proj]",
              relocatable => "true",
            withNotifiers => "true"});
```

ectool

syntax: `ectool export <fileName> [optionals]`

Examples

```
ectool export "c:\ElectricCommanderBackup\Aug 15 2015.xml"

ectool export "c:\ElectricCommanderBackup\Test Proj.xml" --path "/projects[Test Pro
j]"
   --relocatable true --withNotifiers true
```

Back to Top

findObjects

This command returns a sorted list of ElectricFlow 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 ElectricFlow objects and is used by the ElectricFlow web interface to implement the ElectricFlow "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
`objectType`	The type of object to find. **Values include:** `application` `artifact` `artifactVersion` `component` `credential` `directoryProvider` `emailconfig` `emailNotifier` `environment` `formalParameter` `job` `jobStep` `logEntry` `plugin` `procedure` `procedureStep` `process` `processStep` `project` `property` `repository` `resource` `resourcePool` `schedule` `state` `stateDefinition` `transition` `transitionDefinition` `workflow` `workflowDefinition` `workspace` Argument type: String

Arguments	Descriptions

`filters`	(Optional) 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 ElectricFlow 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 ElectricFlow 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)`
	
	Argument type: Collection

`firstResult`	(Optional) The first result to be retrieved, starting from 0 (zero). Argument type: Integer

Arguments	Descriptions
`includeAccess`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, this command also returns access maps for the objects. Argument type: Boolean
`includeEntityRevisions`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, this command also returns the version and/or entity revision if the object is in the search results. Argument type: Boolean
`includeLatestRevision`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, this command also returns the latest revision data for versioned objects. Argument type: Boolean
`maxIds`	(Optional) `<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. Argument type: Integer
`numObjects`	(Optional) `<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. Argument type: Integer
`selects`	(Optional) 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 ElectricFlow Perl API. Argument type: Collection

Arguments	Descriptions
`sorts`	(Optional) This is an ordered list of sort criteria. This option works only when you specify a property name. 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 ElectricFlow Perl API. Argument type: Collection

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. The command can also return full objects from the result list.

ec-perl

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

Example 1

This example shows how to use 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 `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 ElectricFlow:\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 ElectricFlow:\n" . $xPath->findnodes_as_string("/"). "\n";
```

Example 3

This example shows how to make filters with `or` and `and` for finding artifacts matching either of two patterns for the artifact name and `modifyTime` 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 => "2011-11-10T00:00:00.000Z" # 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 => "2011-11-10T00:00:00.000Z" # Arbitrary date
               },
               #filter 2
               {
                   operator => 'or', # apply 'or' for the filters in the list
                     filter => \@artifactNameFilters
               }
           ]
           }
      );
```

Example 4

This example shows how to find a project with a name containing "foo" and with the description "bar".

```
$cmdr->findObjects('project', {
    filter => {operator => 'and',
       filter => [{propertyName => 'projectName',
           operator     => 'contains',
           operand1     => 'foo'},
          {propertyName => 'description',
           operator     => 'equals',
           operand1     => 'bar'}]}});
```

Example 5

This example shows how to find a procedure with the project name "foo" and with the procedure name "bar" or not "bat". (The top level filters are implicitly combined with "and".)

```
$cmdr->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 shows how to find a project with certain property values.

```
$cmdr->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

finishCommand

The agent uses this command to indicate that a command has been run.

Arguments	Descriptions
`agentRequestId`	Request ID of the command. Argument type: String
`status`	Status of the command. Argument type: Integer
`deletedLogFile`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, the agent is deleted or the step log file is not created. Argument type: Boolean
`deletedPostpLogFile`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, the agent is deleted or the postp log file is not created. Argument type: Boolean
`elapsedTime`	(Optional) Elapsed time that the command runs. Argument type: Double
`errorMessage`	(Optional) Error message from the agent. Argument type: String

Arguments	Descriptions
`exit`	(Optional) Exit code of the command. Argument type: Integer
`maxProtocolVersion`	(Optional) Maximum protocol version that the agent supports. Argument type: Integer
`minProtocolVersion`	(Optional) Minimum protocol version that the agent supports. Argument type: Integer
`pingToken`	(Optional) Current ping token. Argument type: Long
`postpExit`	(Optional) Exit code of postp. Argument type: Integer
`protocolVersion`	(Optional) Current agent protocol version. Argument type: Integer
`version`	(Optional) Current agent version. Argument type: String

Positional arguments

`agentRequestId, status`

Response

None or a status OK message.

ec-perl

syntax examples: `$cmdr->finishCommand (<agentRequestId>, <status>, {<optionals>});`

Examples

```
$cmdr->finishCommand ("30f14c6c1fc85cba12bfd093aa8f90e3", 1);
```

ectool

syntax examples: `ectool finishCommand <agentRequestId> <status> [optionals]`

Examples

```
ectool finishCommand "30f14c6c1fc85cba12bfd093aa8f90e3", 1
```

Back to Top

generateDsl

Generates domain-specific language (DSL) script for an existing object.

You must specify the `path` argument.

Arguments	Descriptions
`path`	A property path indicating a single object for which DSL will be generated Argument type: String
`withAcls`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `true`, the generated DSL will include ACLs for the object and the nested objects. Argument type: Boolean

Positional arguments

`path`

Response

Content for the DSL script. For an example, go to Getting Started with DSL on page 836.

ec-perl

syntax:`$cmdr->generateDsl(<path>, {<optionals>});`

Example

```
$cmdr->generateDsl("/resources/local", {withAcls => true});
```

ectool

syntax:`ectool generateDsl <path> [optionals]`

Example

```
ectool generateDsl "/resources/local" --withAcls true
```

Back to Top

getObjects

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.

Arguments	Descriptions
`includeAccess`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, this command also returns access maps for the objects. Argument type: Boolean
`includeLatestRevision`	(Optional) <*Boolean flag* - `0|1|true|false`> If set to `1` or `true`, this command also returns the latest revision data for versioned objects. Argument type: Boolean
`objectIds`	(Optional) 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 ElectricFlow Perl API. Argument Type: Collection
`selects`	(Optional) 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 ElectricFlow Perl API. Argument type: Collection

Positional arguments

None

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 an ElectricFlow 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 ElectricFlow:\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 ElectricFlow:\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

graphStateMachine

Generates a `graph` element with a `stateMachine` DOT graph as CDATA content.

You must specify a `jobId`.

Arguments	Descriptions
`jobId`	The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template. Argument type: UUID
`cluster`	(Optional) <*Boolean flag -* `0|1|true|false`> If this argument is set to `1` or `true`, the graphs are clustered. Argument type: Boolean

Positional arguments

`jobId`

Response

CDATA content.

ec-perl

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

Examples

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

ectool

syntax examples: `ectool graphStateMachine <jobId> [optionals]`

Examples

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

Back to Top

import

Imports data from an XML export file.

You must specify either `file` or `fileName`.

The default timeout is 10800 seconds (180 minutes or 3 hours).

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.

IMPORTANT: When you export a project while a pipeline is in progress, only the full export includes flow runtimes that have been completed. If you want to include in-progress pipeline runs in the path-to-production view and the visual indicators showing their percentage completed in the Release dashboard, set the `excludeJobs` argument to `0` or `false` in the `export` command. When the XML file is imported, the in-progress pipeline runs in the imported project are displayed in the path-to-production view and the Release dashboard. The jobs might be incomplete once the XML is imported.

Arguments	Descriptions
`fileName`	Name of the file to import: <*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. <*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. Argument type: String
`batchSize`	(Optional) <*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 applies only to full import operations. Argument type: Integer
`disableProjectTracking`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true` when importing or exporting a project, even if the original project had Change Tracking enabled, this makes Change Tracking of the newly imported or exported project disabled from its creation. If you do not need to track changes to the new project, this avoids the Change Tracking overhead that would otherwise slow down the import operation, and also saves having to subsequently disable change tracking of the re-imported project. Argument type: Boolean
`disableSchedules`	(Optional) <*Boolean flag* - `0|1|true|false`> If this argument is set to `1` or `true`, imported schedules will be disabled. This argument can modify imported schedules after import and before they are used to start a job. Argument type: Boolean

Arguments	Descriptions
`force`	(Optional) <*Boolean flag -* `0|1|true|false`> If this argument is set to `1` or `true`, an existing object is replaced during a single-object import. This argument can be used to replace a single object if it already exists at the specified property path. Argument type: Boolean
`path`	(Optional) <*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 also used the `path` option. Argument type: String
`preserveId`	(Optional) <*Boolean flag -* `0|1|true|false`> If this argument is set to `1` or `true`, this command tries to preserve the `objectID` during a single-object import. If you are doing a *partial* import, using this option preserves the original job ID or workflow ID. Argument type: Boolean
`reducedDetailChangeHistory`	(Optional) <*Boolean flag -* `0|1|true|false`> Use this argument for large projects containing over 20,000 audited objects with Change Tracking enabled. When this argument is set to `true` or `1`, ElectricFlow automatically decreases the amount of Change History indexing information that it saves in a large project, reducing the level of detail for Change Tracking-intensive operations in the Change History. This can make it harder to revert an object to a specific state and to find information in the Change History when you are troubleshooting or debugging an issue. Set this argument to `false` or `0` to suppress to this behavior so that ElectricFlow does not change the amount of indexing information for a large project. This will cause the operation to take longer and put more load on the database, but the Change History will have the full details of the entities owned by objects in the project. Argument type: Boolean

Positional arguments

`fileName`

Response

None or a status OK message.

ec-perl

syntax examples: `$cmdr->import(<fileName>, {<optionals>});`

```
$cmdr->import({file => <localFileName>, {<optionals>});
```

Examples

```
$cmdr->import("/opt/TestProg.xml");

$cmdr->import({file => "c:\r.xml", path => "/projects[Test]");
```

ectool

syntax examples: `ectool import <remoteFileName> [optionals]`

```
ectool import <localFileName> [optionals]
```

Examples

```
ectool import /mnt/backups/fullBackkup.xml

ectool "c:\project.xml" --path "/projects[Test]"
```

Back to Top

logStatistic

Prints (emits) a statistics value to StatsD.

Arguments	Descriptions
name	The name of the statistic. Argument type: String
type	The type of statistic. Argument type: StatisticType
value	The value of the statistic. Argument type: Long

Positional arguments

`name, type, value`

Response

None or a status OK message.

ec-perl

syntax:`$cmdr->logStatistic (<name>, <type>, <value>);`

Example

```
$cmdr->logStatistic("Interoperability performance test cases", "counters", 7);
```

ectool

syntax:`ectool logStatistic <name> <type> <value>`

Example

```
ectool logStatistic "Interoperability performance test cases" "counters" 7
```

Back to Top

releaseNamedLock

Releases the named lock that synchronizes the name of an object.

Arguments	Descriptions
`lockName`	Name of the lock. Argument type: String
`delay`	(Optional) Number of seconds to delay releasing the lock. Argument type: Integer

Positional arguments

`lockName`

Response

None or a status OK message.

ec-perl

syntax examples: `$cmdr->releaseNamedLock (<lockName>, {<optionals>});`

Examples

```
$cmdr->releaseNamedLock ("Group 1", {delay => 5});
```

ectool

syntax examples: `ectool releaseNamedLock <lockName> [optionals]`

Examples

```
ectool releaseNamedLock "Group 1" --delay 5
```

Back to Top

API Response and Element Glossary

The first part of this topic lists returned some (but not all) of the 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 a * (asterisk) may appear multiple times in a response.

The second part of this Help topic is an element glossary for some (but not all) of the single or "leaf" elements and subcontainer elements. Click here to go to the glossary or notice that each response element is a link—each response element is linked directly to its glossary entry.

access

Contains the set of effective permissions for a user or a group.

Contents:

`changePermissionsPrivilege`

`executePrivilege`

`modifyPrivilege`

`readPrivilege`

aclEntry

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

Contents:

`aclEntryId`

`changePermissionsPrivilege`

`executePrivilege`

`modifyPrivilege`

`readPrivilege`

`principalName`

`principalType`

actualParameter

An `actualParameter` object provides the value for a parameter, which is passed to a procedure when it is invoked.
Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different
from "formal parameters"- formal parameters define the parameters a procedure is expecting, and actual parameters
provide values to use at run-time.

Contents:

actualParameterId

actualParameterName

createTime

modifyTime

value

artifact

Contains elements to define the artifact. An artifact is specified by `groupId` and `artifactKey`. The name of an artifact is in this form "`groupId:artifactKey`". An artifact contains a collection of `artifactVersions`.

Contents:

artifactId

artifactKey

artifactName

artifactVersionNameTemplate

createTime

description

groupId

lastModifiedBy

modifyTime

owner

propertySheetId

artifactVersion

A "concrete" version of an artifact that contains a collection of files stored in the artifact repository.

Contents:

artifactKey

artifactName

artifactVersionId

artifactVersionName

artifactVersionState

buildNumber

createTime

majorMinorPatch

modifyTime

owner

propertySheetId

publisherJobId

publisherJobName

publisherJobStepId

`dependentArtifactVersion`

`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	modifyTime
createTime	name
description	owner
directoryProviderId	position
domainName	propertySheetId
emailAttribute	providerIndex
enableGroups	providerName
fullUserNameAttribute	providerType
groupBase	realm
groupMemberAttributes	url
groupMemberFilter	useSSL
groupNameAttribute	userBase
groupSearchFilter	userNameAttribute
lastModifiedBy	userSearchFilter
managerDn	userSearchSubtree

testDirectoryProvider

Contains the results of testing a directory provider configuration as a list of test result blocks.

Each block contains a result with details about any failures. The `findGroupsTest` block also includes a list of groups for the test user.

The `findUserTest` block includes information about the user or users that matched the test username.

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 ElectricFlow license.

Contents:

createTime

customerName

evaluation

expirationDate

featureName

- gracePeriod
- lastModifiedBy
- licenseId
- modifyTime
- owner
- productName
- property*
- propertySheet*
- signature

licenseUsage

Contains information about ElectricFlow license usage.

Note: Your response will be different depending on how you are licensed for ElectricFlow 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`	`lastRunTime`
`alive`	`modifyTime`
`code`	`owner`
`details`	`pools`
`message`	`port`
`pingToken`	`propertySheetId`
`protocolVersion`	`proxyCustomization`
`state`	`proxyHostName`
`time`	`proxyPort`
`version`	`proxyProtocol`
`artifactCacheDirectory`	`repositoryNames`
`createTime`	`resourceDisabled`
`description`	`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	monthDays
attachedCredentials	owner
beginDate	priority
createTime	procedureName
credentialName	projectName
description	propertySheetId
endDate	scheduleDisabled
interval	scheduleId
intervalUnits	scheduleName
lastModifiedBy	startTime
lastRunTime	stopTime
misfirePolicy	timeZone
modifyTime	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 ElectricFlow 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	subproject
logFileName	timeLimit
modifyTime	timeLimitUnits
owner	workingDirectory
parallel	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 ElectricFlow-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 ElectricFlow-generated ID for this actual parameter object.
`actualParameterName`	string	The name of the parameter. This name is unique within the step, and at runtime 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.
`agentUncPath`	string	UNC path used by Windows ElectricFlow 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 ElectricFlow Web servers to access the workspace.

Returned element	Type	Description/Value
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, recent calls, and 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 ElectricFlow-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 ElectricFlow-generated ID for this artifact version object.
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 ElectricFlow server interprets either name form correctly.

Returned element	Type	Description/Value
`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.
`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 ElectricFlow-generated identifier for this particular call.

Returned element	Type	Description/Value
`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)
`changePermissionsPrivile ge`	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.
`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 ElectricFlow 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.

Returned element	Type	Description/Value
`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 ElectricFlow-generated ID for this credential object.
`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.

Returned element	Type	Description/Value
`customerName`	string	The name of a company and/or group name with a company that is using ElectricFlow.
`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 ElectricFlow 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.
`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).
`dependentArtifactVersions`	list	A list of one or more artifact versions.
`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> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
`destinations`	string	A space-separated list of valid email addresses, email aliases, or ElectricFlow usernames, 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 ElectricFlow-generated ID for this directory provider object.

Returned element	Type	Description/Value
`domainName`	string	The name of the domain from which the Active Directory servers 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 ElectricFlow-generated ID for this email configuration object.
`emailConfigName`	string	The name of the email configuration.
`emailNotifierId`	number	The unique ElectricFlow-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.

Returned element	Type	Description/Value
`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".
`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.

Returned element	Type	Description/Value
`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.
`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).

Returned element	Type	Description/Value
`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 ElectricFlow-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 ElectricFlow license expires.
`groupBase`	string	This string is prepended to the `basedn` to construct the directory DN that contains group records.
`groupId`	number	The unique ElectricFlow-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.

Returned element	Type	Description/Value
`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 ElectricFlow-generated ID for this group settings object.
`hostName`	string	The computer name or IP address for the machine containing the ElectricFlow server or agent.
`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.

Returned element	Type	Description/Value
`interval`	string	The repeat interval for starting new jobs.
`intervalUnits`	enum	Possible values are: `hours|minutes|seconds|continuous` If set to `continuous`, ElectricFlow 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 that is assigned automatically when the job is created. The system 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.
`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 username) 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 ElectricFlow.

Returned element	Type	Description/Value
`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 ElectricFlow-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.
`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 ElectricFlow-generated ID number for the log entry record.
`logFileName`	string	A custom log file name produced by running the step. By default, ElectricFlow 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.

Returned element	Type	Description/Value
`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 "ElectricFlow" - name of the email user on whose behalf ElectricFlow sends email notifications.
`majorMinorPatch`	string	`major.minor.patch` component of the version attribute for the artifact.
`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 ElectricFlow 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.

Returned element	Type	Description/Value
`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.
`mutable`	boolean	If "true," the member list of this group is editable within ElectricFlow 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.

Returned element	Type	Description/Value
`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 (username) 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 username.
`path`	string	The property path that specifies the object to use.
`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 ElectricFlow-generated ID for the plugin object.
`pluginKey`	string	The name of the plugin as displayed on the ElectricFlow 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 ElectricFlow 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 ElectricFlow 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.

Returned element	Type	Description/Value
`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 ElectricFlow 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.
`postProcessor`	string	This program looks at the step output to find errors and warnings. ElectricFlow 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.

Returned element	Type	Description/Value
`procedureId`	number	The unique ElectricFlow-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: ElectricFlow
`project`	name	The name of the project associated with the plugin.
`projectId`	number	The unique ElectricFlow-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 ElectricFlow-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 ElectricFlow Agent used for a proxy resource.

Returned element	Type	Description/Value
`proxyPort`	number	The ElectricFlow 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 ElectricFlow Web Interface at this time.)
`publisherJobId`	number	The ElectricFlow-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 ElectricFlow-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 ElectricFlow-generated ID for the artifact repository.
`repositoryIndex`	integer	The order of the repository within a list of repositories.
`repositoryName`	string	The name of the artifact repository.

Returned element	Type	Description/Value
`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, ElectricFlow will not start new steps on this resource. Defaults to "false".
`resourceId`	number	The unique ElectricFlow-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, ElectricFlow 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.
`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.

Returned element	Type	Description/Value
`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` elements.
`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, ElectricFlow does not start any new jobs from the schedule. Defaults to "false".
`scheduleId`	number	The unique ElectricFlow-generated ID for the schedule.
`scheduleName`	string	The name of the schedule - may be a path to the schedule.
`schemaVersion`	number	The ElectricFlow 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`
`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.

Returned element	Type	Description/Value
`startingState`	string	The initial state of the workflow.
`startTime`	string	Formatted `hh:mm`, using the 24-hour clock. Using this schedule, ElectricFlow starts creating jobs at this time on the specified days.
`stateDefinitionId`	number	The unique ElectricFlow-generated ID for this state definition object.
`stateDefinitionName`	string	The name of the state definition.
`stateId`	number	The unique ElectricFlow-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 ElectricFlow-generated ID for the step.
`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. ElectricFlow stops creating new jobs at this time, but a job in progress will continue to run. If `stopTime` is not specified, ElectricFlow creates one job only on each specified day.
`subject`	string	Refers to the object the event concerns (similar to `container`).

Returned element	Type	Description/Value
`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.
`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.

Returned element	Type	Description/Value
`transitionDefinitionId`	number	The unique ElectricFlow-generated ID for this transition definition.
`transitionDefinitionName`	string	The name of the transition definition.
`transitionId`	number	The unique ElectricFlow-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 ElectricFlow server verifies the agent's identity using SSL certificate verification. • untrusted - the ElectricFlow server does not verify agent identity. Potentially, an untrusted agent is a security risk.
`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/`.

Returned element	Type	Description/Value
`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 ElectricFlow-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.
`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 ElectricFlow-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. Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server.
`value`	string	For a string property, this is the value of the property. For a sheet property, this argument is invalid.

Returned element	Type	Description/Value
`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 ElectricFlow-generated ID for this workflow definition.
`workflowDefinitionName`	string	The name of the workflow definition.
`workflowId`	number	The unique ElectricFlow-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 ElectricFlow 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, ElectricFlow uses the directory it created for the job in the ElectricFlow 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 ElectricFlow-generated ID for the workspace.
`workspaceName`	string	The name of the workspace.

Returned element	Type	Description/Value
`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 ElectricFlow-generated ID for this zone.
`zoneName`	string	The name of the zone.

Using the ElectricFlow RESTful API

The RESTful API is easier to use than the Perl API but more difficult to use than DSL methods to perform ElectricFlow operations such as:

- Create and call procedures
- Model and deploy applications
- Create and manage resources
- Create environment models and add resources to them
- Model and run pipelines
- Model and run releases
- Create and manage artifacts
- Create and manage object properties

To access the RESTful API resources and operations and execute a request, you navigate to the RESTful API URI and enter the appropriate information in the Electric Cloud API UI.

Review these guidelines before using the RESTful API:

- For most RESTful APIs, you enter only the information in the parameters fields to get a response. When you enter information about a resource and its operations in the Electric Cloud API UI, the UI generates a response consisting of
    - **Request URL**–Response in a browser. Go to the URL to see the results.
    - **Response Body**–Response in a standard format with the application level status, the results, and any errors. This is also referred to as the *Response Payload*.
    - **Response Code**–Status of the response at the protocol level.
    - **Response Headers**–Information about the response format.
- However, some APIs have special arguments, such as *key-value pairs*, that are not included in the Request URL response.

  To ensure the special arguments are included in the response, put the code for these arguments in the **body** parameter field. The parameter content type is application/json.
- Batch API is not supported in the RESTful API.
- You can use any web development language that you want with the RESTful APIs.
- When you use a language binding, the response is JSON content that you can use as a hash map. You can use any language binding, such as RubyGems or Python.

- PUT operations
    - ElectricFlow uses the PUT operation to update a specific object.

      In a PUT API call, you specify the values that you want to change.

      The response is that only the values that you specify will change.
    - In ElectricFlow, the PUT operation works like the PATCH API call in the current version of HTTP/REST.
- GET operations

  ElectricFlow uses the GET operation to get basic information about the project, not everything.
    - To get all the projects, the Electric Cloud API UI generates this URL:

      `http://chronic3:8000/rest/v1.0/projects`
    - To get a specific project, the Electric Cloud API UI generates this URL:

      `http://chronic3:8000/rest/v1.0/projects/myPlayground`
    - If the API has a fully qualified property, put an extra slash in the URL:

      `/rest/v1.0/server//foo`

Accessing the RESTful API

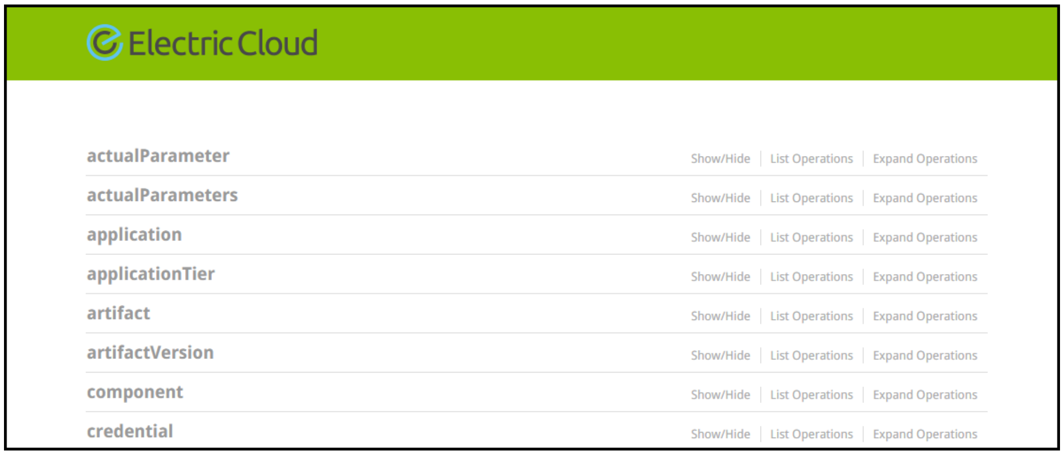
To access the ElectricFlow RESTful API, go to
https://<*ElectricFlow_server_hostname*>/rest/doc/v1.0/
where <*ElectricFlow_server_hostname*> is the hostname or IP address of the ElectricFlow server.

IMPORTANT: The hostname must be the fully qualified domain name (FQDN).

When you enter https://<*ElectricFlow_server_hostname*>/rest/doc/v1.0/, you may not be able to log in directly because of a browser issue, which occurs on browsers such as Internet Explorer, Google Chrome, Mozilla Firefox, and Safari. An error message appears stating that the connection is not private or is untrusted or that there is a problem with the website's security certificate.

Click the appropriate links to continue to the website.

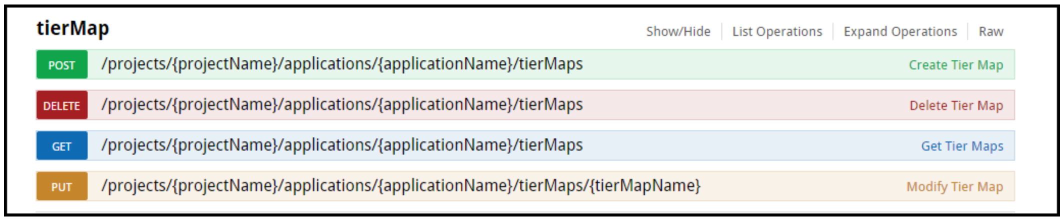
The RESTful API web page opens:


Electric Cloud
actualParameter Show/Hide | List Operations | Expand Operations
actualParameters Show/Hide | List Operations | Expand Operations
application Show/Hide | List Operations | Expand Operations
applicationTier Show/Hide | List Operations | Expand Operations
artifact Show/Hide | List Operations | Expand Operations
artifactVersion Show/Hide | List Operations | Expand Operations
component Show/Hide | List Operations | Expand Operations
credential Show/Hide | List Operations | Expand Operations

Using the RESTful API

In the Electric Cloud API UI page:

1. Select a resource and click on the resource name.



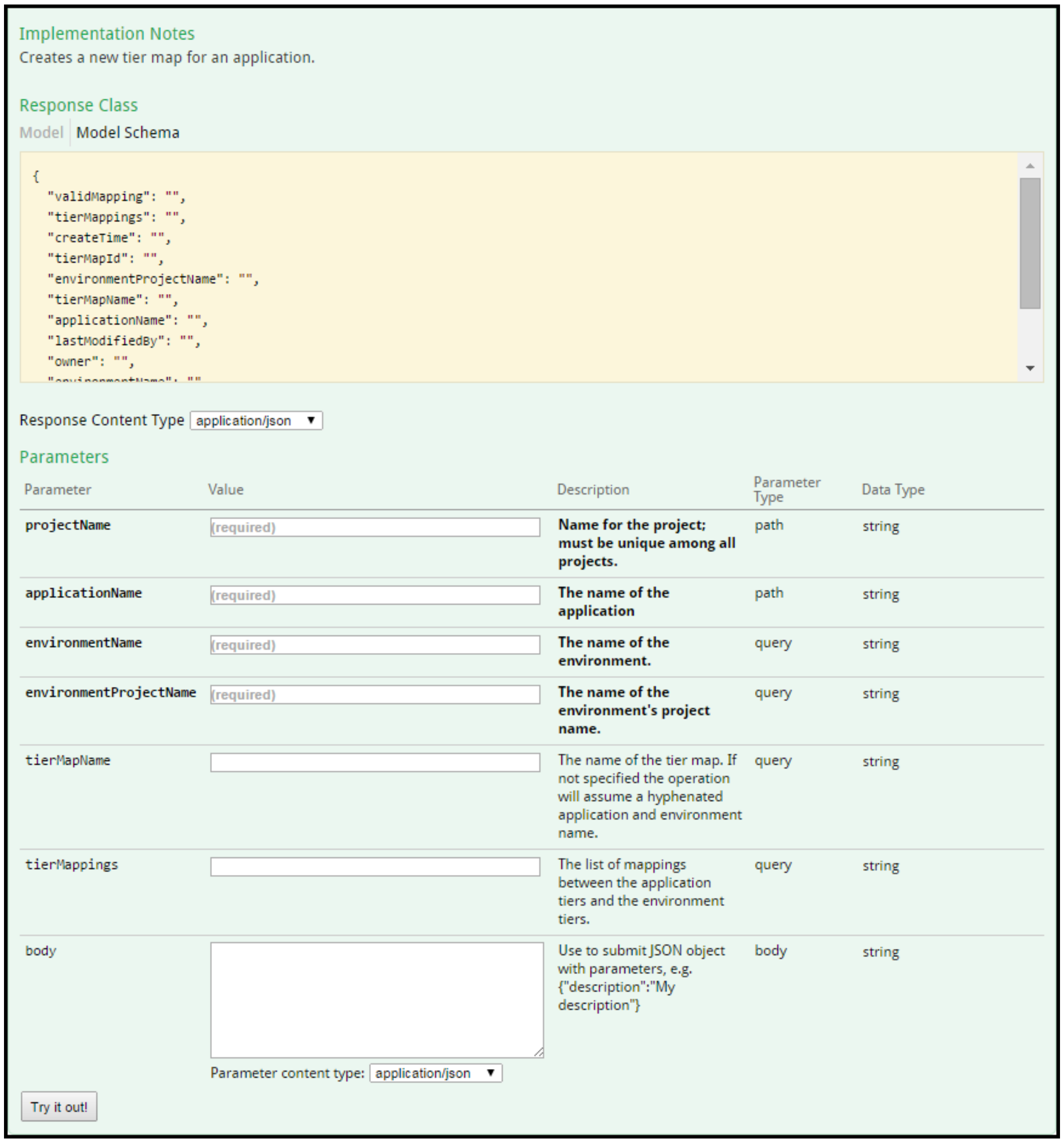
POST /projects/{projectName}/applications/{applicationName}/tierMaps Create Tier Map
DELETE /projects/{projectName}/applications/{applicationName}/tierMaps Delete Tier Map
GET /projects/{projectName}/applications/{applicationName}/tierMaps Get Tier Maps
PUT /projects/{projectName}/applications/{applicationName}/tierMaps/{tierMapName} Modify Tier Map

The operations for the resource appear below it.

2. Select an operation and click on the operation name.

   Parameters (referred to as arguments in the ElectricFlow API), for the operation appears.

   When you select POST, response information appears.



Implementation Notes

Creates a new tier map for an application.

Response Class

Model | Model Schema

```
{
  "validMapping": "",
  "tierMappings": "",
  "createTime": "",
  "tierMapId": "",
  "environmentProjectName": "",
  "tierMapName": "",
  "applicationName": "",
  "lastModifiedBy": "",
  "owner": "",
```

Response Content Type application/json

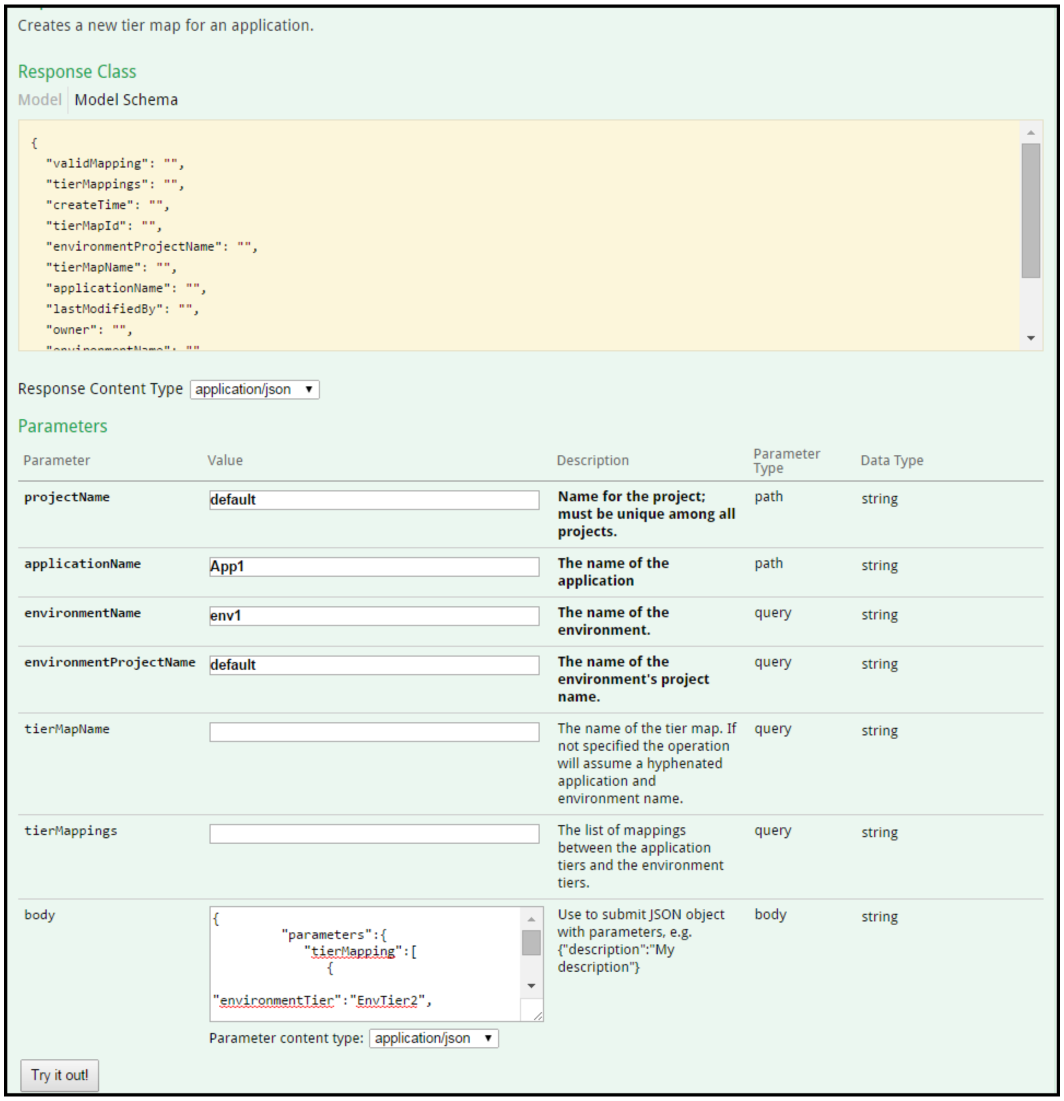
Parameters

Parameter	Value	Description	Parameter Type	Data Type
projectName	(required)	**Name for the project; must be unique among all projects.**	path	string
applicationName	(required)	**The name of the application**	path	string
environmentName	(required)	**The name of the environment.**	query	string
environmentProjectName	(required)	**The name of the environment's project name.**	query	string
tierMapName		The name of the tier map. If not specified the operation will assume a hyphenated application and environment name.	query	string
tierMappings		The list of mappings between the application tiers and the environment tiers.	query	string
body	Parameter content type: application/json	Use to submit JSON object with parameters, e.g. {"description":"My description"}	body	string

3. Enter the appropriate information in the fields.

   If the operation has special arguments that the Electric Cloud API UI does not include in the Request URL, you also enter code in the **body** field to ensure that the arguments are in the RESTful API response.

   In this example, the POST operation for tierMap, which is the same as the **createTierMap** API command, includes key-value pairs that the Electric Cloud API UI does not include in the Request URL.



Creates a new tier map for an application.

Response Class

Model | Model Schema

```
{
  "validMapping": "",
  "tierMappings": "",
  "createTime": "",
  "tierMapId": "",
  "environmentProjectName": "",
  "tierMapName": "",
  "applicationName": "",
  "lastModifiedBy": "",
  "owner": "",
```

Response Content Type: application/json

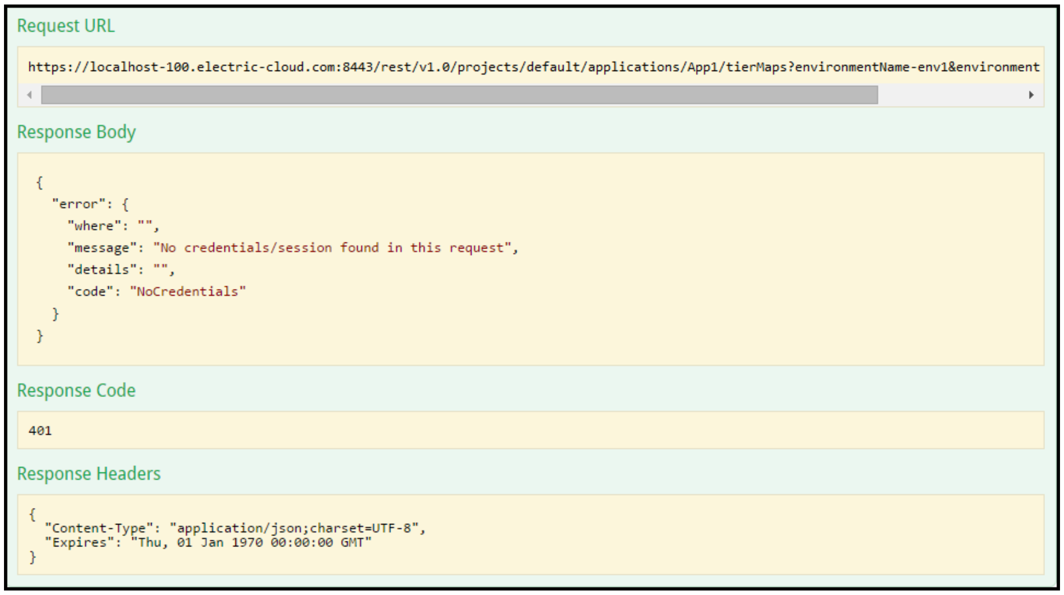
Parameters

Parameter	Value	Description	Parameter Type	Data Type
projectName	default	Name for the project; must be unique among all projects.	path	string
applicationName	App1	The name of the application	path	string
environmentName	env1	The name of the environment.	query	string
environmentProjectName	default	The name of the environment's project name.	query	string
tierMapName		The name of the tier map. If not specified the operation will assume a hyphenated application and environment name.	query	string
tierMappings		The list of mappings between the application tiers and the environment tiers.	query	string
body	`{ "parameters":{ "tierMapping":[ { "environmentTier":"EnvTier2",`	Use to submit JSON object with parameters, e.g. {"description":"My description"}	body	string

Parameter content type: application/json

Try it out!

4. Click **Try it out!** to run the resource.



Request URL

```
https://localhost-100.electric-cloud.com:8443/rest/v1.0/projects/default/applications/App1/tierMaps?environmentName-env1&environment
```

Response Body

```
{
  "error": {
    "where": "",
    "message": "No credentials/session found in this request",
    "details": "",
    "code": "NoCredentials"
  }
}
```

Response Code

```
401
```

Response Headers

```
{
  "Content-Type": "application/json;charset=UTF-8",
  "Expires": "Thu, 01 Jan 1970 00:00:00 GMT"
}
```

The response has this information:

- **Request URL**–Response in a browser. Go to the URL to see the results.
- **Response Body**–Response in a standard format with the application level status, the results, and any errors. This is also referred to as the *Response Payload*.
- **Response Code**–Status of the response at the protocol level.
- **Response Headers**–Information about the response format.

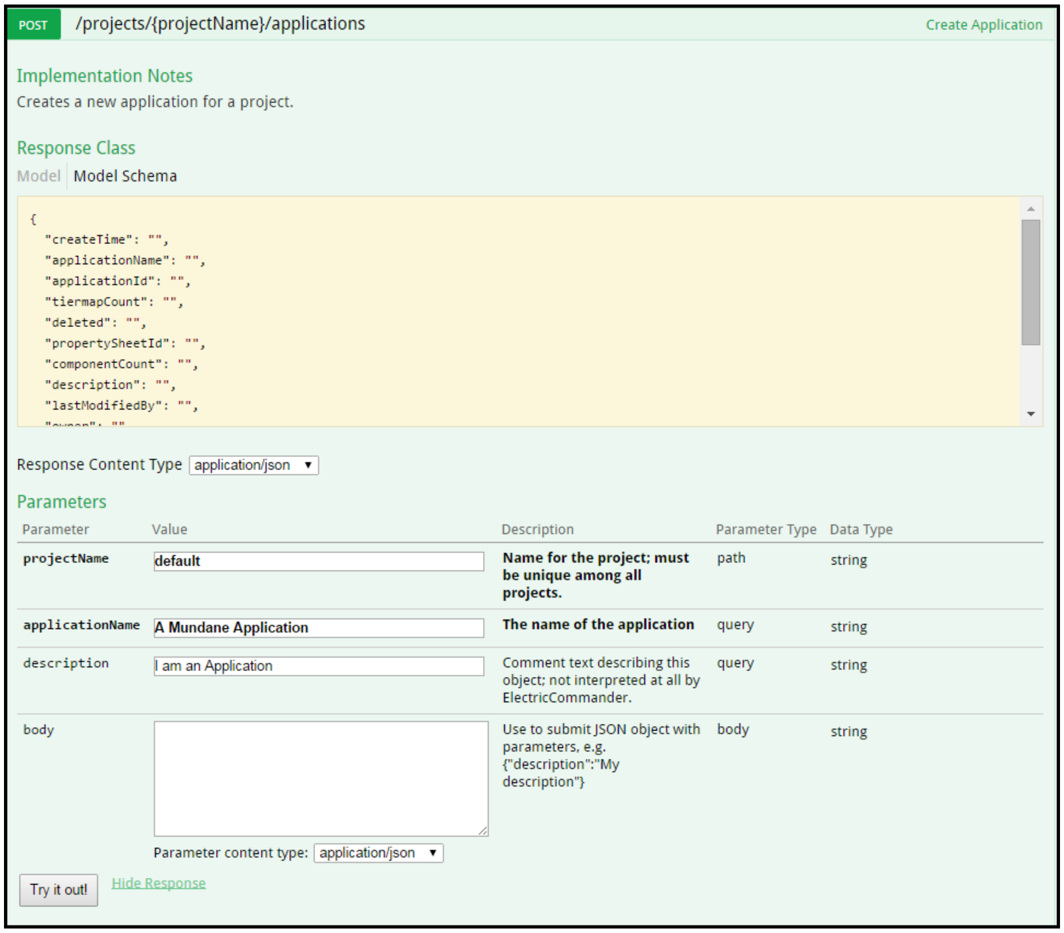
RESTful API Examples

In the ElectricFlow implementation of RESTful APIs, POST APIs are used to create new objects and are not independent.

The following examples show how to create objects without submitting a JSON object with parameters.

POST Operations Without Special Arguments

To design a new application, enter the values in the parameter section as follows:



POST /projects/{projectName}/applications Create Application

Implementation Notes

Creates a new application for a project.

Response Class

Model | Model Schema

```
{
  "createTime": "",
  "applicationName": "",
  "applicationId": "",
  "tiermapCount": "",
  "deleted": "",
  "propertySheetId": "",
  "componentCount": "",
  "description": "",
  "lastModifiedBy": "",
```

Response Content Type application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
projectName	default	**Name for the project; must be unique among all projects.**	path	string
applicationName	A Mundane Application	**The name of the application**	query	string
description	I am an Application	Comment text describing this object; not interpreted at all by ElectricCommander.	query	string
body		Use to submit JSON object with parameters, e.g. {"description":"My description"}	body	string

Parameter content type: application/json

Try it out! Hide Response

After clicking **Try it out!**, the response is a Request URL:

```
https://localhost-100.electric-cloud.com/rest/v1.0/projects/default/applications?appli
cationName=A%20Mundane%20Application&description=I%20am%20an%20Application
```

To create an application process in an application, enter values for the following parameters:

- **projectName**: default
- **applicationName** : A Mundane Application
- **processName**: AppProc1
- **timeLimit**: 30
- **timeLimitUnits**: seconds

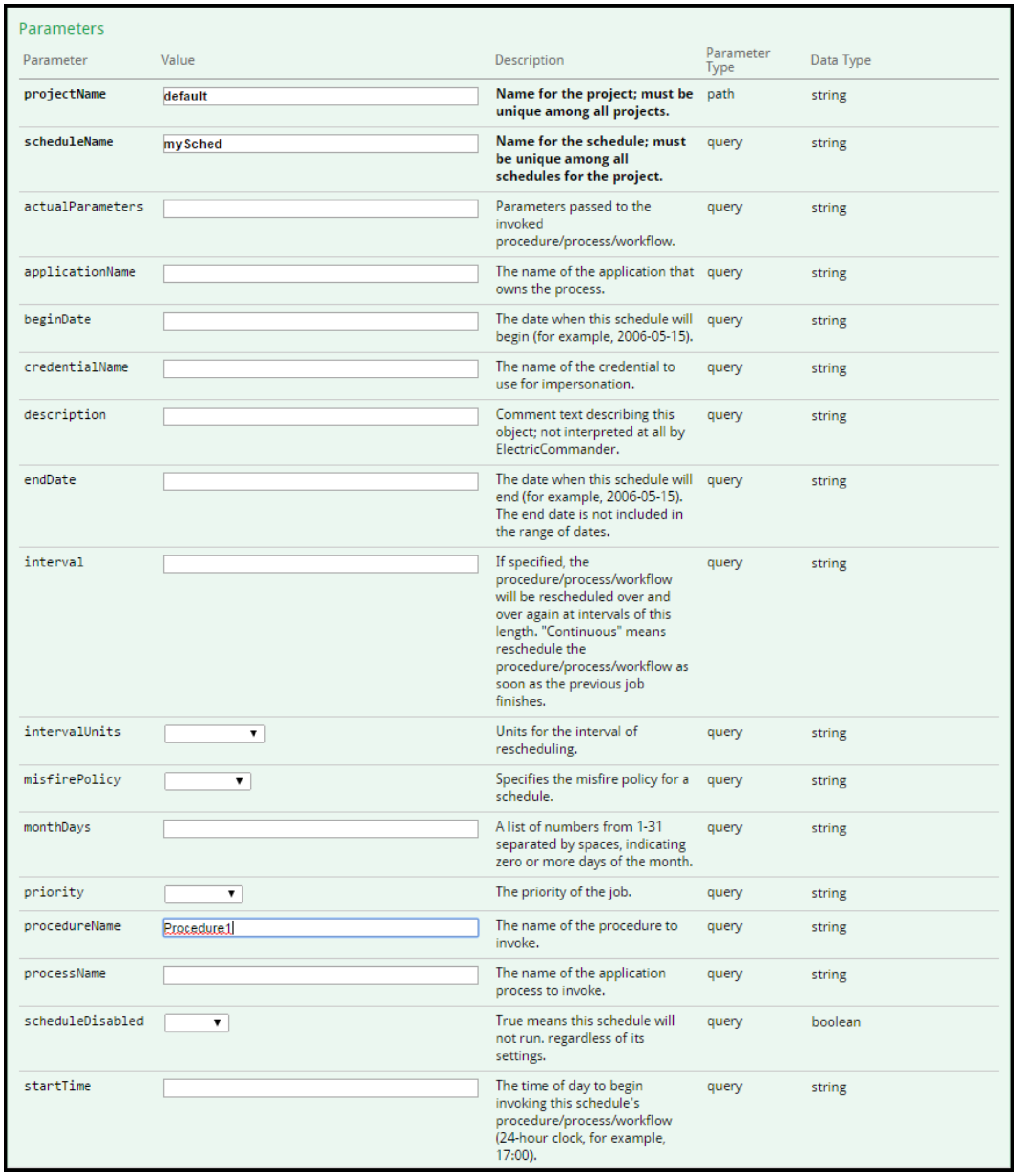
The response is this Request URL:

```
https://flow-demo-uat.electric-cloud.com/rest/v1.0/projects/default/applications/A%20M
undane%20Application/processes?processName=AppProc1&timeLimit=30&timeLimitUnits=second
s
```

POST Operations with Special Arguments

When APIs have special arguments, such as key-value pairs, you have to put code for them in the **body** parameter field.

To make a schedule with actual parameters, enter the values in the parameter section as shown below:

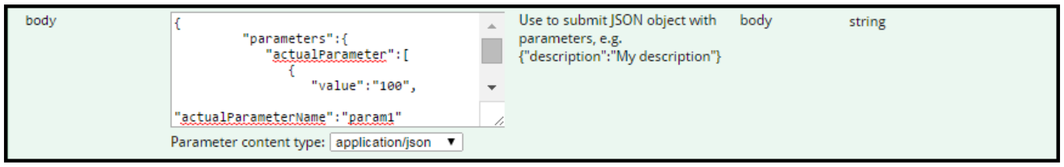


Parameters

Parameter	Value	Description	Parameter Type	Data Type
projectName	default	**Name for the project; must be unique among all projects.**	path	string
scheduleName	mySched	**Name for the schedule; must be unique among all schedules for the project.**	query	string
actualParameters		Parameters passed to the invoked procedure/process/workflow.	query	string
applicationName		The name of the application that owns the process.	query	string
beginDate		The date when this schedule will begin (for example, 2006-05-15).	query	string
credentialName		The name of the credential to use for impersonation.	query	string
description		Comment text describing this object; not interpreted at all by ElectricCommander.	query	string
endDate		The date when this schedule will end (for example, 2006-05-15). The end date is not included in the range of dates.	query	string
interval		If specified, the procedure/process/workflow will be rescheduled over and over again at intervals of this length. "Continuous" means reschedule the procedure/process/workflow as soon as the previous job finishes.	query	string
intervalUnits		Units for the interval of rescheduling.	query	string
misfirePolicy		Specifies the misfire policy for a schedule.	query	string
monthDays		A list of numbers from 1-31 separated by spaces, indicating zero or more days of the month.	query	string
priority		The priority of the job.	query	string
procedureName	Procedure1	The name of the procedure to invoke.	query	string
processName		The name of the application process to invoke.	query	string
scheduleDisabled		True means this schedule will not run. regardless of its settings.	query	boolean
startTime		The time of day to begin invoking this schedule's procedure/process/workflow (24-hour clock, for example, 17:00).	query	string

You also enter the following in the body parameter field:

```
{
          "parameters":{
             "actualParameter":[
                {
                   "value":"100",
                    "actualParameterName":"param1"
                }
            ]
        }
}
```



body	`{ "parameters":{ "actualParameter":[ { "value":"100", "actualParameterName":"param1"`	Use to submit JSON object with parameters, e.g. {"description":"My description"}	body	string

Parameter content type: application/json

The response is this Request URL:

```
https://localhost-100.electric-cloud.com/rest/v1.0/projects/default/schedules?schedule
Name=mySched&procedureName=Procedure1
```

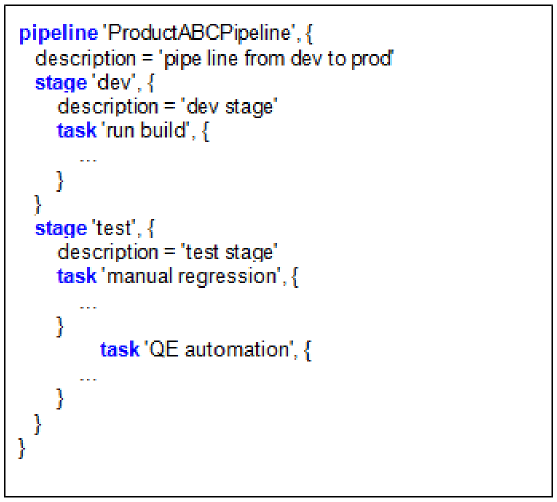
To make a tier map with tier mappings, the `tierMapping` argument in the `createTierMap` API requires a key-value pair.

Using the ElectricFlow DSL

The ElectricFlow domain-specific language (DSL) is an intuitive, easy-to-use computer language that allows you to model continuous delivery (CD) and Application Release Automation (ARA) solutions. It has these capabilities:

- Easy to understand
- Self documenting
- Process-as-code: You can define your processes as high-level code that can be versioned, tested, and reused.
- User defined versions for any object
- Modeling of applications, pipelines, and releases using simple, logical constructs
- Based on the Groovy language
- Supports various means of code reuse
- Access to all the ElectricFlow API commands

This is a snippet of a DSL script:



```
pipeline 'ProductABCPipeline', {
  description = 'pipe line from dev to prod'
  stage 'dev', {
     description = 'dev stage'
     task 'run build', {
        ...
     }
  }
  stage 'test', {
     description = 'test stage'
     task 'manual regression', {
        ...
     }
           task 'QE automation', {
        ...
     }
  }
}
```

Why Use DSL?

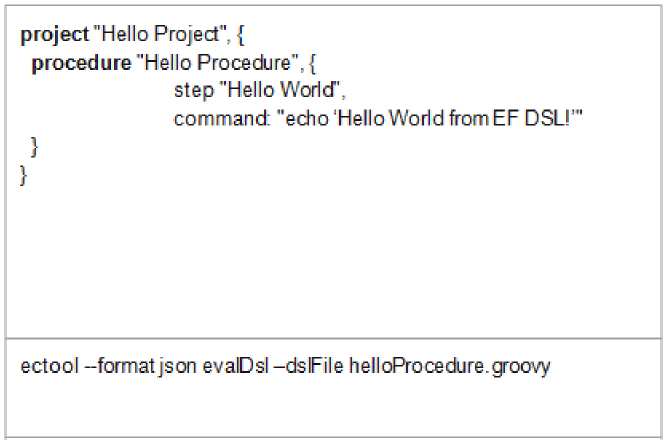
ElectricFlow DSL is the easiest of the ElectricFlow programming constructs to use.

For example, this is a comparison of DSL (the easiest to use) and `ec-perl` (the most difficult to use):

DSL	ec-perl
Declarative (what)	Script-based, imperative (how)
Concise	Relatively verbose

DSL	ec-perl
Do not need to understand Groovy to read DSL	Need to understand Perl
The entire DSL script is sent only once to the server for evaluation	Client-server chattiness: Each API invocation is separate call to the server

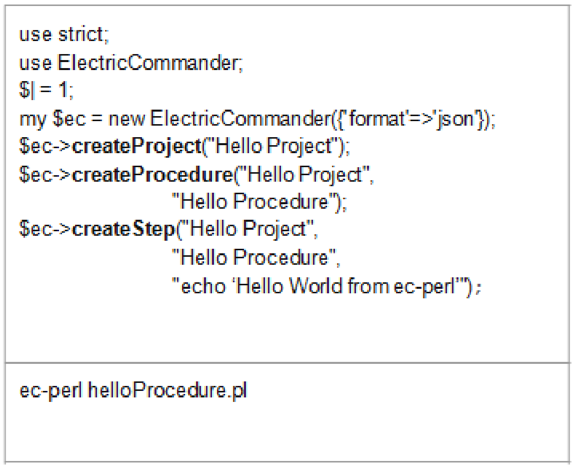
This is the DSL version of the helloProcedure in JSON:



```
project "Hello Project", {
  procedure "Hello Procedure", {
                    step "Hello World",
                    command: "echo 'Hello World from EF DSL!'"
  }
}
```

```
ectool --format json evalDsl –dslFile helloProcedure.groovy
```

This is the ec-perl version of the helloProcedure:



```
use strict;
use ElectricCommander;
$| = 1;
my $ec = new ElectricCommander({'format'=>'json'});
$ec->createProject("Hello Project");
$ec->createProcedure("Hello Project",
                    "Hello Procedure");
$ec->createStep("Hello Project",
                    "Hello Procedure",
                    "echo 'Hello World from ec-perl'");
```

```
ec-perl helloProcedure.pl
```

You create models or scripts to represent the ElectricFlow objects using the DSL methods and let the ElectricFlow DSL engine take care of invoking the correct API to create or update the object if it already exists. The DSL is based on Groovy scripting language; however, you do not need to know Groovy in detail to use it.

The benefits of using DSL scripts are:

- You can create one master model or script and run it one or more times for different scenarios to build, test, and deploy your software or application by passing different parameter values for evaluating the script.
- You can automate software delivery processes to produce repeatable, scalable, and efficient results.
- You can quickly and easily create and deploy applications using DSL scripts. Using these scripts enables a higher-order command-line interface (CLI) with richer syntax than the CLI on your system.
- Users with little to no programming experience levels can create and run DSL to perform specific operations. They do not need to know how to use the ElectricFlow API or UI to create and deploy an application.
- Using DSL scripts make Continuous Delivery and Continuous Integration possible at scale.

Using DSL for Object Versioning

In addition to all of the previous benefits, DSL also provides the crucial functionality of *object versioning*. You can easily generate a DSL file for any ElectricFlow object either by using the API or UI for the Deploy and Release objects. See Generating DSL Scripts on page 843 for more information on how to generate DSL scripts.

These DSL files can then be used for further editing or stored in a version management system of your choice. DSL is a great way for creating and using user-defined versions of any ElectricFlow object.

Running DSL

You can run DSL one of these ways:

- From the command-line interface
- Using REST
- From a step command
- Using the DSL IDE. Go to DSL IDE on page 844 for the details.

From the Command-Line Interface

Use the `evalDsl` command to evaluate and run a DSL script. See the `evalDsl` command in the "API Commands - Miscellaneous Management" section of the ElectricFlow API Guide for a list of the supported arguments.

These `ectool` examples show how to evaluate a DSL script.

- When specifying the DSL text, enter `ectool evalDsl <dsl>`. For example:

  ```
  ectool evalDsl "project 'My WAR file' "
  ```

- When specifying the path to the DSL script on the client, enter `evalDsl --dslFile <dslFile>`. For example:

  ```
  ectool evalDsl --dslFile c:/dslScripts/myWarFile.dsl
  ```

Using REST

Go to `http://<ElectricFlow_server>:8000/rest/v1.0/server/dsl`, where `<ElectricFlow_server>` is the hostname (the fully qualified domain name [FQDN]) or IP address of the ElectricFlow server.

1. Select the **dsl** method.
2. For the Request, select or enter the following information for the Request and then click **Try it out!**.

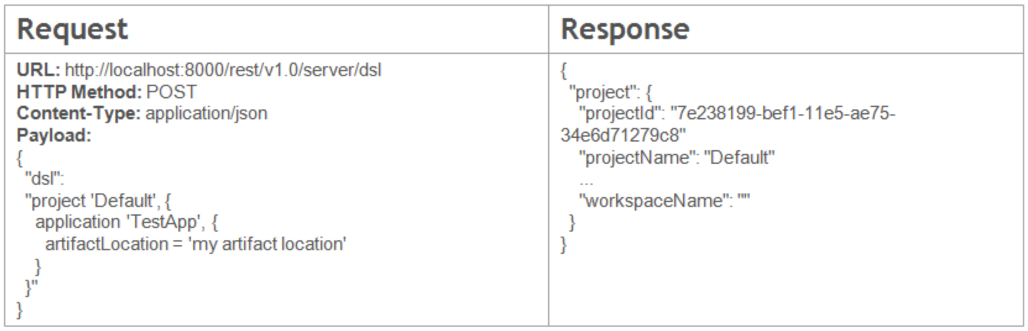
   For **Response Content Type**, select **application/json**.

   For **Project**, enter **Default**.

   For **Application**, enter **TestApp**.

   For **artifactLocation**, enter **my artifact location**.

   The response appears in the **Response Body** field and includes the project ID, project name, and workspace name.

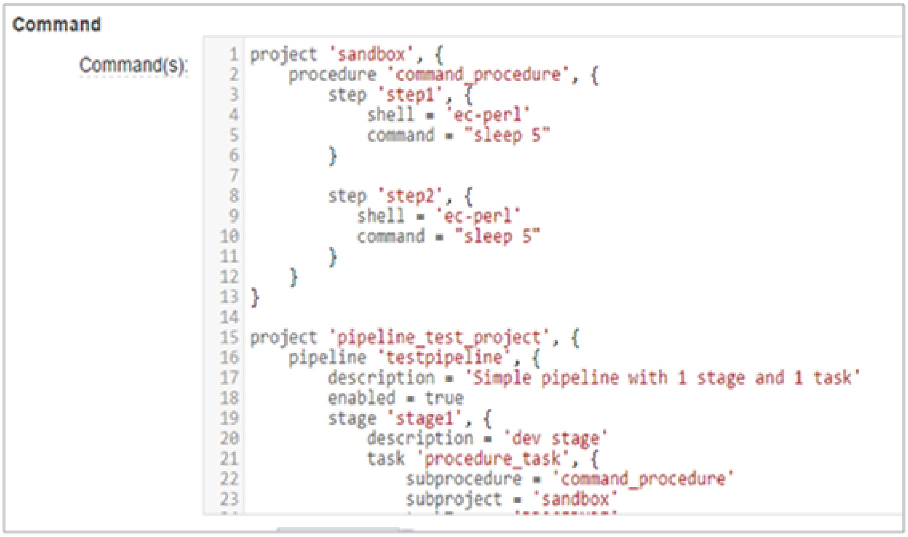


Request	Response

URL: http://localhost:8000/rest/v1.0/server/dsl
HTTP Method: POST
Content-Type: application/json
Payload:
{
"dsl":
"project 'Default', {
application 'TestApp', {
artifactLocation = 'my artifact location'
}
}"
}
	{
	"project": {
	"projectId": "7e238199-bef1-11e5-ae75-34e6d71279c8"
	"projectName": "Default"
	...
	"workspaceName": ""
	}
	}

From a Step Command

You can enter the DSL script as a step command in a procedure.



```
project 'sandbox', {
    procedure 'command_procedure', {
        step 'step1', {
            shell = 'ec-perl'
            command = "sleep 5"
        }

        step 'step2', {
           shell = 'ec-perl'
           command = "sleep 5"
        }
    }
}

project 'pipeline_test_project', {
    pipeline 'testpipeline', {
        description = 'Simple pipeline with 1 stage and 1 task'
        enabled = true
        stage 'stage1', {
            description = 'dev stage'
            task 'procedure_task', {
                subprocedure = 'command_procedure'
                subproject = 'sandbox'
```

You can use `ectool` to evaluate and run this DSL script by entering `ectool evalDsl --dslFile {0}`.

Modeling with DSL

The DSL provides an alternative to the UI for authoring and managing ElectricFlow objects. It allows you to perform all the same operations as the RESTful APIs or Perl APIs but much easier and simpler.

- Create and call procedures
- Model and deploy applications
- Create and manage resources
- Create environment models and add resources to them
- Model and run pipelines
- Model and run releases
- Create and manage artifacts
- Create and manage object properties

For more information about the ElectricFlow DSL, go to:

- ElectricFlow DSL: Process-As-Code for All Your Automation Needs
- Process as Code: An Introduction to the ElectricFlow Automation DSL

Getting Started with DSL

Getting Help on DSL Methods

To get information on the DSL methods, use the `evalDsl --describe` argument as follows:

- Obtaining the complete list of DSL methods for ElectricFlow objects:

```
ectool evalDsl dsl --describe 1
```

- Obtaining DSL method details for a specific ElectricFlow object, such as an application:

```
ectool evalDsl application --describe 1
```

- Obtaining details for a specific API, such as `getProcedure`:

  ```
  ectool evalDsl getProcedure --describe 1
  ```

Creating and Running DSL Scripts

Using ectool

To create and run a DSL script using ectool:

1. Use a text editor or Groovy IDE to create a script using ElectricFlow DSL methods, and save it.
2. Log into `ectool`.
3. Enter `evalDsl` to run the DSL script.

   You can use these arguments with `evalDsl`.

   - `debug`—ElectricFlow generates debug output as the DSL script is evaluated when the `debug` argument is set to `1` or `true`.
   - `describe`—ElectricFlow prints a description of the DSL text when the `describe` argument is set to `1` or `true`.
   - `parameters`—Parameters are passed to the script by ElectricFlow as JSON text.

Using ec-perl

To create and run a DSL script using ec-perl:

1. Use any text editor or Groovy IDE to create a script using ElectricFlow DSL methods, and save it.
2. If you did *not* use `ec-perl` in for the previous step, start `ec-perl`.
3. Enter `evalDsl` to run the DSL script.

   You can use these arguments to get more information about results as the script runs.

   - `debug`—ElectricFlow generates debug output as the DSL script is evaluated when the `debug` argument is set to `1` or `true`.
   - `describe`—ElectricFlow prints a description of the DSL text when the `describe` argument is set to `1` or `true`.
   - `parameters`—Parameters passed to the script by ElectricFlow as JSON text.

Common Use Cases

Generating a DSL Script for an Existing ElectricFlow Object

To generate a DSL script for an existing ElectricFlow object, which was created through a Perl API, a RESTful API, or the UI, enter

```
ectool generateDsl <path>
```

where *<path>* is the path to the ElectricFlow object for which you want to generate the DSL script.

For example, if you have a resource named *local* in your ElectricFlow instance:

1. Run the following command to redirect the output to a file (for example, myScript.dsl):

   ```
   ectool generateDsl /resources/local> myScript.dsl
   ```

This command generates output redirected to the specified file, which looks similar to the following:

```
resource 'local', {
  description = 'Local resource created during installation.'
  artifactCacheDirectory = null
  hostName = '192.168.10.10'
  hostType = 'CONCURRENT'
  port = '7800'
  proxyCustomization = null
  proxyHostName = null
  proxyProtocol = null
  repositoryNames = null
  resourceDisabled = '0'
  shell = null
  trusted = '0'
  useSSL = '1'
  workspaceName = null
  zoneName = 'default'
}
```

2. Use the script file created in the previous steps with the `evalDsl` command to create or update the resource in ElectricFlow.

   You can also edit the file to add or update resource attributes before using the script with `evalDsl`.

   ```
   ectool evalDsl --dslFile myScript.dsl
   ```

Passing Parameters or Arguments to Your DSL Script

ElectricFlow DSL lets you create a template script using script parameters instead of hard-coding all the values in the script. You can then invoke the same script with different parameter values each time to create different ElectricFlow object instances. For example, you could use the following script to create a resource that uses SSL in the *secure* zone:

```
zone 'secure'
  resource() {
  resourceName = args.resourceName
  hostName = args.resourceIP
  hostType = 'CONCURRENT'
  resourceDisabled = '0'
  trusted = '1'
  useSSL = '1'
  zoneName = 'secure'
}
```

The script has the values `args.resourceName` and `args.resourceIP` for the `resourceName` and `hostName` resource attributes respectively. These argument or parameter values can be passed from the command line to the DSL script in JSON form using the following command:

For Linux:

```
ectool evalDsl --dslFile myScript.dsl --parameters '{"resourceName":"MyFirstResource",
"resourceIP":"192.168.10.12"}'
```

For Windows:

```
ectool evalDsl --dslFile myScript.dsl --parameters "{\"resourceName\":\"MyFirstResourc
e\", \"resourceIP\":\"192.168.10.12\"}"
```

Note the special handling required on Windows for passing command-line arguments that contain double quotes and spaces. To allow spaces and other special characters in a command-line argument, Windows requires wrapping the value in quotes. Also, if the value itself contains quotes, you must escape those quotes by using a backslash '\'. For alternate method for passing parameter values for DSL, see Passing Parameters or Arguments to Your DSL Script using a File on page 839.

Passing Parameters or Arguments to Your DSL Script using a File

Parameters to a DSL script can be passed using a file that contains the parameters in JSON format as follows:

```
ectool evalDsl --dslFile myScript.dsl --parametersFile myParams.json
```

Where the file `myParams.json` may contain:

```
{
  "resourceName" : "MyFirstResource",
  "resourceIP" : "192.168.10.12"
}
```

Go to Passing Parameters or Arguments to Your DSL Script on page 838for details about using parameters in a DSL script.

Evaluating Your DSL Script in a Specific Application Context

ElectricFlow DSL supports a simple and intuitive nested structure to represent the logical structure of ElectricFlow objects. This allows the DSL methods to be evaluated in a specific context, such as with respect to an ElectricFlow project, application, or pipeline.

For example, the following is a very simple script that can create an application with an application tier in a project:

```
// ElectricFlow DSL engine will create project 'Default' unless it already exists
project ('Default')  {
  //'Deploy world' application will be created within 'Default' project unless it alre
ady exists
  application('Deploy world') {
    //'Web Tier' application tier will be created within 'Deploy world' application un
less it already exists
    applicationTier('Web Tier')
  }
}
```

Using ElectricFlow APIs in DSL

All ElectricFlow APIs available through ec-perl are available for use in your DSL script with the exception of `publishArtifactVersion` and `retrieveArtifactVersions`. The syntax for invoking an ElectricFlow API in DSL is as follows:

```
<methodName> (argumentName1: value1, argumentName2:value2, ... )
```

For example, the `getProcedure` API can be invoked as:

```
def proc = getProcedure (procedureName: 'RunInstances', projectName: 'DeployUtilitie
s')
```

See the ElectricFlow Perl API Commands on page 29 for the complete list of API commands.

Understanding Transactions in DSL

By default, all operations invoked in a single DSL script run in one transaction. This means that if an error is encountered in the script, the entire script is rolled back. For example, in the following script:

```
// -- Transaction begins here
project ('MyTestProject')
application('MyTestApp') {
  applicationTier('Tier1') {
    process('Deploy') {
      processStep('step1',
        applicationTierName: 'Tier1',
        processStepType: 'command',
        errorHandling: 'failProcedure',
        subproject: '/plugins/EC-Core/project',
        subprocedure: 'RunCommand',
        actualParameter: [ shellToUse: 'sh', commandToRun: 'echo hi' ])
    }
  }
}
// -- Transaction ends here
```

if an error is encountered while the script creates the process step `step1` after creating the process, then the application tier, the application, then the entire transaction is rolled back to avoid a partially created application, application tier, and process. However, as with other ElectricFlow APIs, if the system considers the error to be *retryable* (such as a database locking error), then the entire DSL is re-evaluated.

When using certain ElectricFlow APIs in the DSL script, you must run them in a separate transaction. To do so, enclose the required part of the script in a *transaction*. For example, if you request a procedure to start in the DSL script and want to monitor its progress in the same script, you must commit the request in a separate transaction before monitoring begins. This lets the system pick up the request for execution. The following example shows how to do so:

```
// -- Transaction one begins here
procedure projectName: ‘Hello Project’, procedureName: ‘testRunProcedure’
def resp

// -- Transaction one ends here
// -- Transaction two begins here

transaction
{

// calling the runProcedure in its own transaction
// so that the procedure kicks off when this script is
// running and we can monitor its progress.

  resp=runProcedure(
    projectName: 'Hello Project',
    procedureName: 'testRunProcedure',
    actualParameter: [
      friend: 'James',
    ]
  )
}
// -- Transaction two ends here
// -- Transaction three begins here

// Now let's grab the jobId launched to run the procedure
def id=resp.jobId

// Let's wait for it to finish
def String status=''
```

```
while(status != "completed") {
    status=getProperty(propertyName: 'status', jobId: id).value;
    println "status: $status"
    sleep (1000)
}

// -- Transaction three ends here

def resp

// Calling the runProcedure in it's own transaction
// so that the procedure kicks off when this script is
// running and we can monitor its progress.
transaction{
  resp=runProcedure(
    projectName: 'Hello Project',
    procedureName: 'testRunProcedure',
    actualParameter: [
      friend: 'James',
    ]
  )
}

// Now let's grab the jobId launched to run the procedure
def id=resp.jobId

// Let's wait for it to finish
def String status=''
while(status != "completed") {
    status=getProperty(propertyName: 'status', jobId: id).value;
    println "status: $status"
    sleep (1000)
}
```

The above script has three transactions:

1. Before the transaction block
2. The transaction block itself within which the procedure run request happens
3. After the transaction block where it polls for the procedure to complete

When evaluating such a DSL script, whether the script is retried after a retryable error depends on where the error occurs. If it occurs before the transaction block is encountered, then the script is retried. But if it occurs after the transaction block is encountered, then the script is not retried.

Adding Support for Augmenting the Groovy Runtime classpath for ectool evalDsl

You can use external .jar files with DSL. ElectricFlow DSL is based on Groovy so that you can take advantage of all Groovy and Java capabilities. You can use any Groovy or Java libraries in your DSL script. The libraries can be made available to the ElectricFlow DSL runtime engine when the DSL script is executed using the `serverLibraryPath` parameter.

This example from the *HttpBuilder—Easy HTTP client for Groovy's* wiki shows how to use the HTTPBuilder classes in a DSL script:

1. Create a file named `httputil.groovy` with the following content. The script uses the `groovyx.net.http.HTTPBuilder` class to make an HTTP GET request.

   ```
   import groovyx.net.http.HTTPBuilder
   import static groovyx.net.http.Method.GET
   import static groovyx.net.http.ContentType.JSON

   def result = [:]
   def http = new HTTPBuilder( 'http://ajax.googleapis.com' )
   http.request( GET, JSON ) {
     uri.path = '/ajax/services/search/web'
     uri.query = [ v:'1.0', q: 'Calvin and Hobbes' ]

     response.success = { resp, json -&gt;
       assert json.size() == 3
       json.responseData.results.each {
         result.put(it.titleNoFormatting, it.visibleUrl)
       }
     }
   }
   result
   ```

2. Create a directory named `/opt/dslSamples/lib` and make it accessible from the ElectricFlow server with the following .jar files:

   ```
   http-builder-0.6

   json-lib-2.3-jdk15.jar

   xml-resolver-1.2.jar
   ```

3. Evaluate script using the following command:

   ```
   ectool evalDsl --dslFile httputil.groovy --serverLibraryPath /opt/dslSamples/lib
   ```

   When the DSL is evaluated on the ElectricFlow server, any .jar files contained in the directory specified for `serverLibraryPath` will be available to the ElectricFlow DSL runtime engine.

Any Groovy files and Java class files contained in the directory specified for `serverLibraryPath` will also be available to the script evaluated by `evalDsl`. For example, if the directory contains a Groovy class `my.sample.dsl.DslUtil` in the directory structure `my/sample/dsl/DslUtil.groovy`, the script can use the Groovy class by importing the class in the script as with any other class.

Debugging Your DSL Script

You can use debug logging as well as the `println` method for debugging DSL scripts.

Using Debug Logging

You can use the `debug` argument to return debug messages in the `evalDsl` response for your script while it is evaluated by the ElectricFlow DSL engine. These messages are useful for debugging DSL scripts. For example:

```
ectool evalDsl --dslFile myScript.dsl --debug 1
```

or

```
ectool evalDsl "project 'My WAR file'" --debug 1
```

Using the println Method

You can use the standard Groovy `println` method in your DSL script to print messages to the client console while it is evaluated by the ElectricFlow DSL engine. The line numbers for lines that produced `println` output are included. For example, entering `ectool evalDsl --dslFile myfile.dsl`, where `myfile.dsl` contains:

```
def result
for (int i =0; i< 10; i++) {
   println 'Creating project: ' + 'ABTest-' + i
   result = project 'ABTest-' + i, resourceName: 'res1'
}
result
```

returns the following output:

```
<response requestId="1" nodeId="10.0.1.15">
    <value>
Result: project[name=ABTest-9,id=7777e4ff-7941-11e6-94c1-34e6d71279c8]
Console output:
Line 0003: [OUT] Creating project: ABTest-0
Line 0003: [OUT] Creating project: ABTest-1
Line 0003: [OUT] Creating project: ABTest-2
Line 0003: [OUT] Creating project: ABTest-3
Line 0003: [OUT] Creating project: ABTest-4
Line 0003: [OUT] Creating project: ABTest-5
Line 0003: [OUT] Creating project: ABTest-6
Line 0003: [OUT] Creating project: ABTest-7
Line 0003: [OUT] Creating project: ABTest-8
Line 0003: [OUT] Creating project: ABTest-9
</value>  </response>
```

If you also use the `debug` argument , the `println` output is interleaved with the debug logging to the console. The line numbers for lines that produced `println` output and debug logging are included. The `println` output is similarly interleaved with debug logging for exceptions where the debug logs are returned in the evalDsl response. For example, entering `ectool evalDsl --dslFile myfile_that_throws_exception.dsl --debug 1`, where `myfile_that_throws_exception.dsl` contains:

```
project 'printingInInvalidDsl'
println("Causing NPE now..")
def a
a.name //will cause NPE
println("Should not print")
```

returns the following output:

```
ectool error [InvalidScript]: Unknown exception during DSL eval at line 4: Cannot get
property 'name' on null object
Line 0001: DSL method 'project'
Line 0001: Checking if project exists with args {projectName=printingInInvalidDsl}
Line 0001: project exists: false
Line 0001: Invoking 'createProject' with args {projectName=printingInInvalidDsl}
Line 0001: Done invoking 'createProject'
Line 0002: [OUT] Causing NPE now..
Details:
Cannot get property 'name' on null object
```

Generating DSL Scripts

There are two ways to generate DSL scripts:

- Through ectool
- Through the ElectricFlow UI

Through ectool

Use the `ectool generateDsl <path> [optionals]` API command to generate an XML file representing the contents of an ElectricFlow object, such as a project, procedure, application, environment, master component, environment or resource template, pipeline, or release. Go to the generateDsl on page 765 command for the full syntax and for ectool examples.

Through the ElectricFlow UI

To quickly and easily generate and export a DSL script from the ElectricFlow UI, click the button in the following UIs and select the **DSL export** option:

- Visual editors for Applications, Environments, Master Components, Environment and Resource Templates, and Pipelines
- Release Definition

For an example showing how to export DSL scripts from the UI, go to DSL IDE on page 844.

DSL IDE

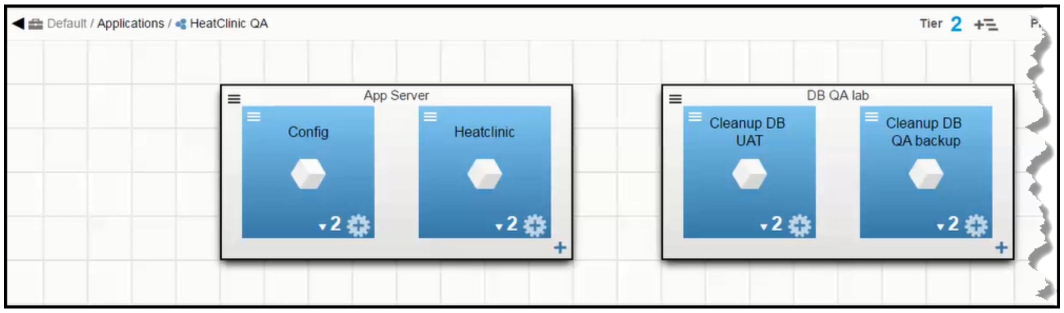
The modeling process is simplified by using the DSL IDE because it allows you to run DSL code from the platform UI rather than having to go to an external IDE or command-line interface to run the `evalDsl` API call. The DSL IDE is an ElectricFlow plugin that can be invoked from the ElectricFlow platform UI.

The following example shows how you can generate and export a DSL script for an existing object in the UI (an application), modify it, and run it through the DSL IDE to apply your changes. For instructions to use the DSL IDE, go to the EC-DSLIDE plugin Help file.

1. Navigate to an object in the UI.

Example:

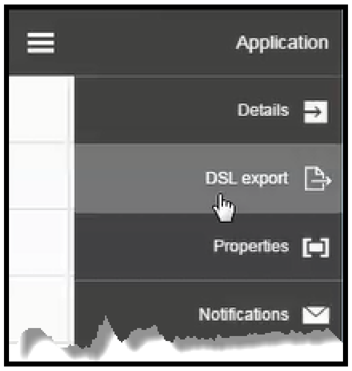
Go to an application called "HeatClinic QA" and view it in the Applications Visual Editor.



2. Click the ☰ button for that object and select the **DSL export** option to generate a DSL script and export it.

   ## Example:

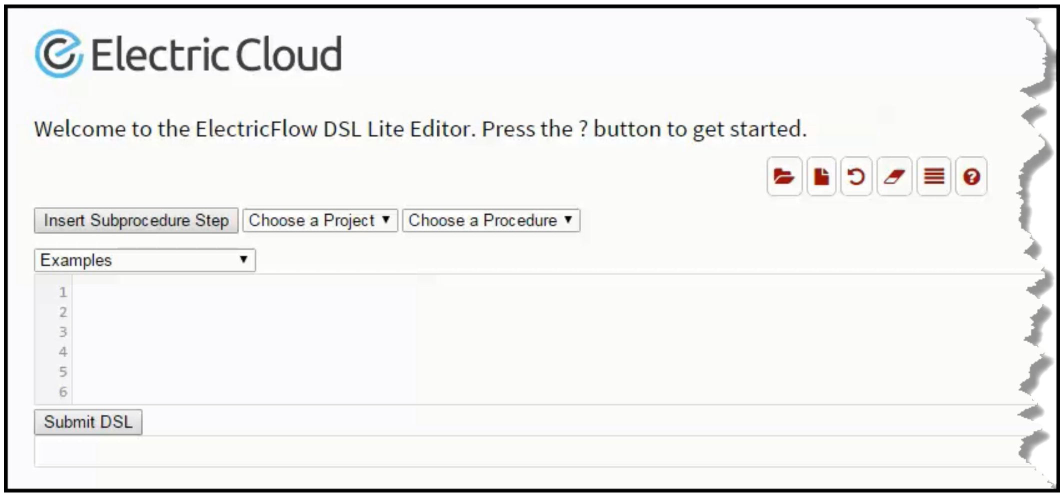
   Click the ☰ button in the upper right corner of the Applications Visual Editor.

3. Click **DSL IDE** in the platform UI to invoke the DSL IDE in ElectricFlow.

   The DSL IDE opens.

   ## Example:

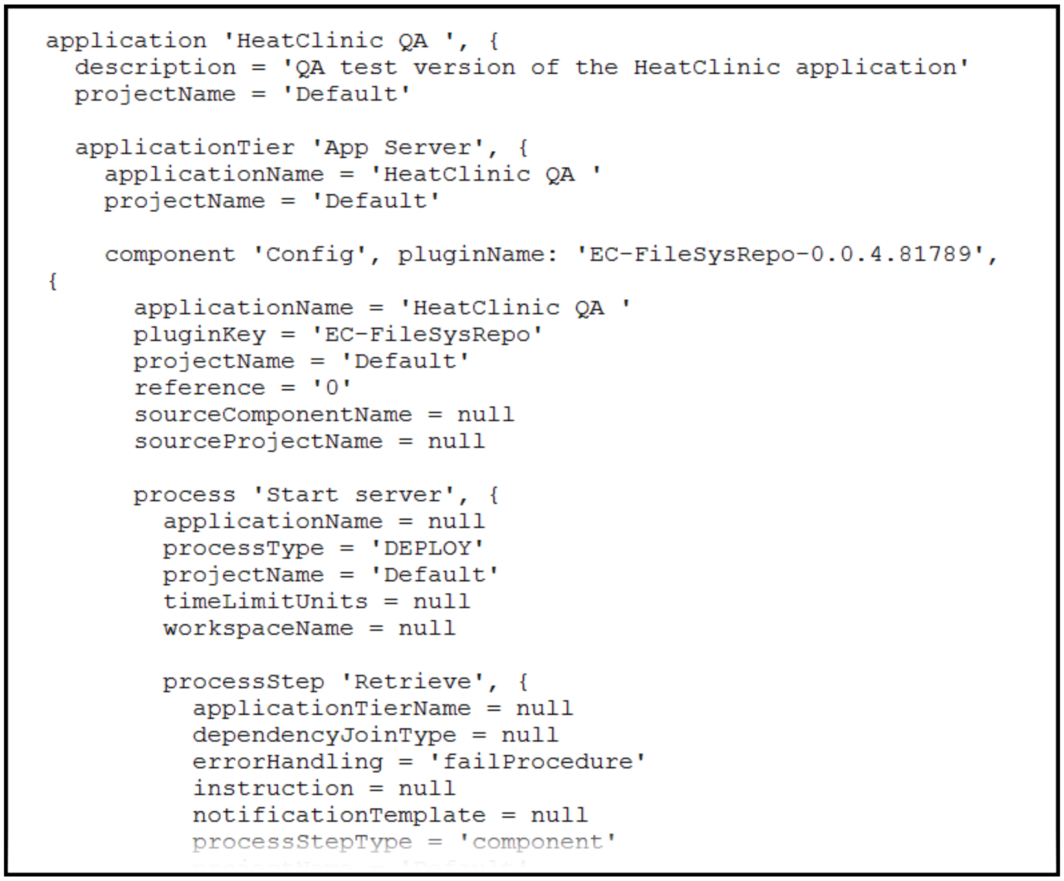
   

   Welcome to the ElectricFlow DSL Lite Editor. Press the ? button to get started.

4. Copy the content from the DSL script and paste it into DSL IDE editor window.

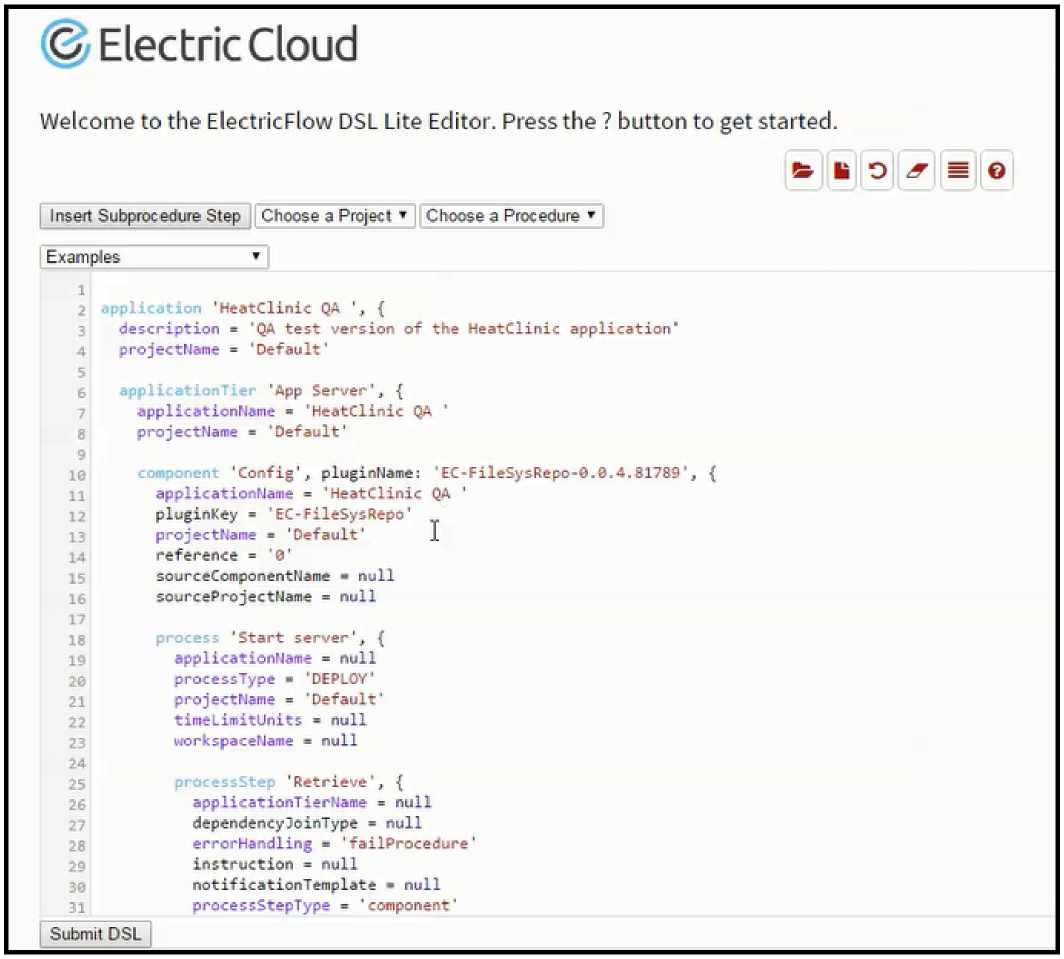
Example:

This is a snippet of the DSL script that was exported:



```
application 'HeatClinic QA ', {
  description = 'QA test version of the HeatClinic application'
  projectName = 'Default'

  applicationTier 'App Server', {
    applicationName = 'HeatClinic QA '
    projectName = 'Default'

    component 'Config', pluginName: 'EC-FileSysRepo-0.0.4.81789',
{
      applicationName = 'HeatClinic QA '
      pluginKey = 'EC-FileSysRepo'
      projectName = 'Default'
      reference = '0'
      sourceComponentName = null
      sourceProjectName = null

      process 'Start server', {
        applicationName = null
        processType = 'DEPLOY'
        projectName = 'Default'
        timeLimitUnits = null
        workspaceName = null

        processStep 'Retrieve', {
          applicationTierName = null
          dependencyJoinType = null
          errorHandling = 'failProcedure'
          instruction = null
          notificationTemplate = null
          processStepType = 'component'
```

Copy and paste the content into the DSL IDE:



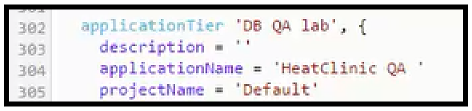
Welcome to the ElectricFlow DSL Lite Editor. Press the ? button to get started.

```

application 'HeatClinic QA ', {
  description = 'QA test version of the HeatClinic application'
  projectName = 'Default'

  applicationTier 'App Server', {
    applicationName = 'HeatClinic QA '
    projectName = 'Default'

    component 'Config', pluginName: 'EC-FileSysRepo-0.0.4.81789', {
      applicationName = 'HeatClinic QA '
      pluginKey = 'EC-FileSysRepo'
      projectName = 'Default'
      reference = '0'
      sourceComponentName = null
      sourceProjectName = null

      process 'Start server', {
        applicationName = null
        processType = 'DEPLOY'
        projectName = 'Default'
        timeLimitUnits = null
        workspaceName = null

        processStep 'Retrieve', {
          applicationTierName = null
          dependencyJoinType = null
          errorHandling = 'failProcedure'
          instruction = null
          notificationTemplate = null
          processStepType = 'component'
```

5. Modify the object.

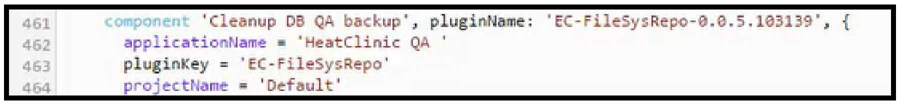
Example:

Change the name of the application tier from "DB QA lab" to "DB QA":



```
  applicationTier 'DB QA lab', {
    description = ''
    applicationName = 'HeatClinic QA '
    projectName = 'Default'
```

Go to the "Cleanup DB QA backup" component in this application tier, and change the component name from "Cleanup DB QA backup" to "Cleanup DB QA".

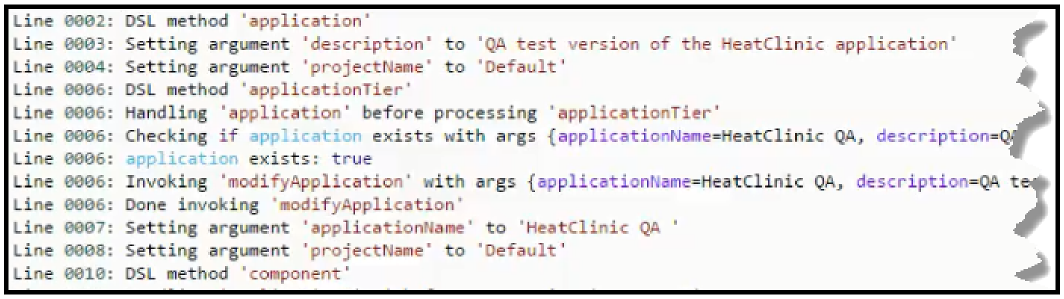


```
    component 'Cleanup DB QA backup', pluginName: 'EC-FileSysRepo-0.0.5.103139', {
      applicationName = 'HeatClinic QA '
      pluginKey = 'EC-FileSysRepo'
      projectName = 'Default'
```

6. Click **Submit DSL**.

   The IDE processes the script line by line.

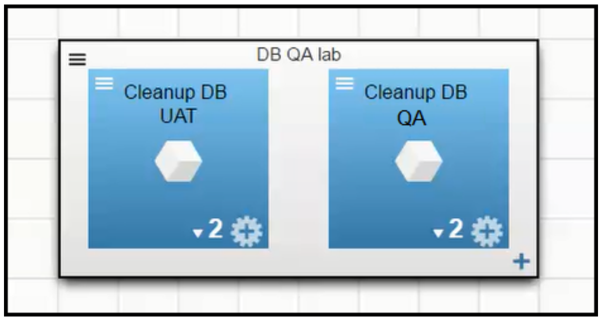
Example:



```
Line 0002: DSL method 'application'
Line 0003: Setting argument 'description' to 'QA test version of the HeatClinic application'
Line 0004: Setting argument 'projectName' to 'Default'
Line 0006: DSL method 'applicationTier'
Line 0006: Handling 'application' before processing 'applicationTier'
Line 0006: Checking if application exists with args {applicationName=HeatClinic QA, description=Q
Line 0006: application exists: true
Line 0006: Invoking 'modifyApplication' with args {applicationName=HeatClinic QA, description=QA te
Line 0006: Done invoking 'modifyApplication'
Line 0007: Setting argument 'applicationName' to 'HeatClinic QA '
Line 0008: Setting argument 'projectName' to 'Default'
Line 0010: DSL method 'component'
```

7. Go to the UI page in the first step to view the changes you made.

Example:



DSL Methods

ElectricFlow supports these DSL methods:

Name	Description
acl	An access control list that controls access to domain objects.
aclEntry	An individual access control list entry (ACE) that allows or denies a privilege on a domain object.
actualParameter	A name/value pair that is passed to a procedure when it is invoked.
application	An ElectricFlow object that you model and deploy to build, test , deploy, and release your software for continuous delivery.
applicationTier	A logical grouping of a components that are part of an application and the resources on which they should be deployed.

Name	Description
`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.
`artifactVersion`	An artifact version is a collection of `0` to `N` files that were published to an artifact repository.
`component`	An object that is based on a specific version of an artifact and is defined in an application.
`credential`	A credential is an object that stores a username and password for later use. You can use credentials for user impersonation and saving passwords for use inside steps. It is usually used in an agent context to authenticate with a third-party system. The password value is not available in the web context for security reasons, so this object has no `getPassword` method.
`deployerApplication`	An application that will be deployed as part of a Deployer in a Release
`deployerConfiguration`	The configuration for an application that will be deployed in a Release
`directoryProvider`	Contains information about the configuration used to communicate with an external directory service (LDAP or ActiveDirectory).
`emailConfig`	Encapsulates all of the mail server configuration information necessary for the ElectricFlow server to send an e-mail message.
`emailNotifier`	E-mail notification to be sent when a particular event occurs.
`environment`	The environment to which an application is deployed.
`environmentTemplate`	A template defining an environment that can be spun up when the application is deployed.
`environmentTemplateTier`	A logical grouping of resources in an environment template.
`environmentTemplateTierMap`	A map that contains the mapping of application tiers to the corresponding environment template tiers.
`environmentTier`	A logical grouping of resources in an environment.
`formalParameter`	An unbound parameter defined on a procedure, workflow definition, and so on.
`gateway`	A secure connection between two zones for sharing or transferring information between the zones.

Name	Description
group	A group of users.
hook	A resource template hook that stores a reference to a procedure in an ElectricFlow project or plugin project. When a resource template is used to create a resource pool, these procedures are invoked.
job	An instance of a procedure run.
jobStep	A step in a job.
license	License data in XML format that describes the usage to which you are entitled.
note	Notes for a Release.
pipeline	An ElectricFlow object that orchestrates a deployment or automation. ElectricFlow supports these types of pipelines: • Release pipeline–Only for an application release. • Generic pipeline-For any deployment or automation.
plugin	An add-on program used by ElectricFlow to integrate with third-party tools, custom dashboards, and unique user experiences based on roles.
procedure	A container for steps that execute a task.
process	An application or component process.
processDependency	A dependency between process steps.
processStep	A step in an application or component process.
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.
property	A `name-value` pair associated with ElectricFlow objects to provide additional information beyond what is already built into the system. Built-in data is also accessible through the property mechanism. ElectricFlow supports intrinsic and custom properties.
release	A ElectricFlow object defined by a pipeline and a Deployer task.

Name	Description
`repository`	An object that stores artifact versions. It primarily contains information about how to connect to a particular artifact repository.
`resource`	An agent machine that is configured to communicate with ElectricFlow and where job steps can be executed.
`resourcePool`	A collection of resources with an ordering policy.
`resourceTemplate`	A template with the required information to provision and later spin up cloud resources on an on-demand basis.
`schedule`	An object that launches a procedure at a specified time in the future, possibly on a regular interval.
`snapshot`	A version of an application with specific artifact versions and at a specific state at any point in time.
`stage`	A logical grouping of pipeline tasks.
`stateDefinition`	A state definition in a workflow definition. Each workflow can contain one or more states.
`step`	A unit of logic that will execute on an agent.
`task`	A representation of task in a stage or gate.
`tierMap`	A map that contains the mapping of application tiers to the corresponding environment tiers.
`transitionDefinition`	How a workflow transitions from one state to another.
`user`	A user defines an account used to log into the system and control access to ElectricFlow objects.
`workflowDefinition`	Workflow objects are split into two types: *Definition* objects and *Instance* objects. Definition objects provide the template for a running workflow instance.
`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.
`zone`	A zone or top-level network created as a way to partition a collection of agents to secure them from use by other groups.

acl

Creates or modifies an access control list that controls access to domain objects.

Required Arguments

None.

Optional Arguments

Name	Description
`applicationName`	The name of the application container of the property sheet that owns the property.
`applicationTierName`	The name of the application tier container of the property sheet that owns the property.
`artifactName`	The name of the artifact container of the property sheet that owns the property.
`artifactVersionName`	The name of the `artifactVersion` container of the property sheet that owns the property.
`componentName`	The name of the component container of the property sheet that 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 that owns the property.
`environmentName`	The name of the environment container of the property sheet that owns the property.
`environmentTemplateName`	The name of the environment template container of the property sheet that owns the property.
`environmentTemplateTierName`	The name of the environment template tier container of the property sheet that owns the property.
`environmentTierName`	The name of the environment tier container of the property sheet that owns the property.
`executePrivilege`	Determines whether the principal can invoke this object as part of a job; this privilege is only relevant for a few objects such as procedures and procedure steps.
`flowName`	The name of the flow container of the property sheet that owns the property.

Name	Description
flowRuntimeName	The name of the flow runtime container of the property sheet that owns the property.
flowRuntimeStateName	The name of the flow runtime state container of the property sheet that owns the property.
flowStateName	The name of the flow state container of the property sheet that owns the property.
flowTransitionName	The name of the flow transition container of the property sheet that 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.
jobId	The primary key or name of the job container of the property sheet that owns the property.
jobStepId	The primary key of the job-step container of the property sheet that owns the property.
notifierName	The name of the notifier container of the property sheet that owns the property.
objectId	The object id as returned by `findObjects`.
path	Property path string.
pipelineName	The name of the pipeline container of the property sheet that owns the property.
pluginName	The name of the plugin container of the property sheet that owns the property.
procedureName	The name of the procedure container of the property sheet that owns the property.
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 that owns the property.
propertySheetId	The primary key of the property sheet that owns the property.

Name	Description
`releaseName`	The name of the Release container of the property sheet that owns the property.
`repositoryName`	The name of the repository container of the property sheet that owns the property.
`resourceName`	The name of the resource container of the property sheet that owns the property.
`resourcePoolName`	The name of the resource pool container of the property sheet that owns the property.
`resourceTemplateName`	The name of the resource template container of the property sheet that owns the property.
`scheduleName`	The name of the schedule container of the property sheet.
`snapshotName`	The name of the snapshot container of the property sheet that owns the property.
`stageName`	The name of the stage container of the property sheet that owns the property.
`stateDefinitionName`	The name of the state definition container of the property sheet that owns the property.
`stateName`	The name of the state container of the property sheet that owns the property.
`stepName`	The name of the step container of the property sheet that owns the property.
`systemObjectName`	The system object.
`taskName`	The name of the task that owns property sheet.
`transitionDefinitionName`	The name of the transition definition container of the property sheet that owns the property.
`transitionName`	The name of the transition container of the property sheet that owns the property.
`userName`	The name of the user container of the property sheet that owns the property.
`workflowDefinitionName`	The name of the workflow definition container of the property sheet that owns the property.

Name	Description
`workflowName`	The name of the workflow container of the property sheet that owns the property.
`workspaceName`	The name of the workspace container of the property sheet.
`zoneName`	The name of the zone container of the property sheet.

DSL Methods for ElectricFlow Objects That Can be Nested Inside

- aclEntry

aclEntry

Creates or modifies an individual access control list entry (ACE) that allows or denies a privilege on a domain object.

Required Arguments

Name	Description
`principalName`	Name of the user or group for this access control entry.
`principalType`	Type of principal for this access control entry (user or group).
`objectType`	The type of object.

Optional Arguments

Name	Description
`applicationName`	The name of the application container of the property sheet that owns the property.
`applicationTierName`	The name of the application tier container of the property sheet that owns the property.
`artifactName`	The name of the artifact container of the property sheet that owns the property.
`artifactVersionName`	The name of the `artifactVersion` container of the property sheet that owns the property.
`changePermissionsPrivilege`	Determines whether the principal can modify access control for the object.

Name	Description
`componentName`	The name of the component container of the property sheet that 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 that owns the property.
`environmentName`	The name of the environment container of the property sheet that owns the property.
`environmentTemplateName`	The name of the environment template container of the property sheet that owns the property.
`environmentTemplateTierName`	The name of the environment template tier container of the property sheet that owns the property.
`environmentTierName`	The name of the environment tier container of the property sheet that owns the property.
`executePrivilege`	Determines whether the principal can invoke this object as part of a job; this privilege is only relevant for a few objects such as procedures and procedure steps.
`flowName`	The name of the flow container of the property sheet that owns the property.
`flowRuntimeName`	The name of the flow runtime container of the property sheet that owns the property.
`flowRuntimeStateName`	The name of the flow runtime state container of the property sheet that owns the property.
`flowStateName`	The name of the flow state container of the property sheet that owns the property.
`flowTransitionName`	The name of the flow transition container of the property sheet that 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.
`jobId`	The primary key or name of the job container of the property sheet that owns the property.
`jobStepId`	The primary key of the job-step container of the property sheet that owns the property.

Name	Description
`modifyPrivilege`	Determines whether the principal can change the contents of the object.
`notifierName`	The name of the notifier container of the property sheet that owns the property.
`objectId`	The object id as returned by `findObjects`.
`path`	Property path string.
`pipelineName`	The name of the pipeline container of the property sheet that owns the property.
`pluginName`	The name of the plugin container of the property sheet that owns the property.
`procedureName`	The name of the procedure container of the property sheet that owns the property.
`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 that owns the property.
`propertySheetId`	The primary key of the property sheet that owns the property.
`readPrivilege`	Determines whether the principal can examine the contents of the object.
`releaseName`	The name of the Release container of the property sheet that owns the property.
`repositoryName`	The name of the repository container of the property sheet that owns the property.
`resourceName`	The name of the resource container of the property sheet that owns the property.
`resourcePoolName`	The name of the resource pool container of the property sheet that owns the property.
`resourceTemplateName`	The name of the resource template container of the property sheet that owns the property.
`scheduleName`	The name of the schedule container of the property sheet.

Name	Description
snapshotName	The name of the snapshot container of the property sheet that owns the property.
stageName	The name of the stage container of the property sheet that owns the property.
stateDefinitionName	The name of the state definition container of the property sheet that owns the property.
stateName	The name of the state container of the property sheet that owns the property.
stepName	The name of the step container of the property sheet that owns the property.
systemObjectName	The system object.
taskName	The name of the task that owns property sheet.
transitionDefinitionName	The name of the transition definition container of the property sheet that owns the property.
transitionName	The name of the transition container of the property sheet that owns the property.
userName	The name of the user container of the property sheet that owns the property.
workflowDefinitionName	The name of the workflow definition container of the property sheet that owns the property.
workflowName	The name of the workflow container of the property sheet that owns the property.
workspaceName	The name of the workspace container of the property sheet.
zoneName	The name of the zone container of the property sheet.

actualParameter

Creates or modifies a name/value pair that is passed to a procedure when it is invoked.

Required Arguments

Name	Description
`actualParameterName`	The name of the parameter to create, modify, or delete.
`projectName`	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`applicationName`	The name of the application, if the actual parameter is on an application process step.
`componentName`	The name of the component, if the actual parameter is on a component process step.
`flowName`	The name of the flow to which the flow state belongs.
`flowStateName`	The name of the flow state,if the formal parameter is in a flow state.
`newName`	New name for an existing object that is being renamed.
`procedureName`	The name of the procedure.
`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.
`releaseName`	The name of the Release container of the property sheet that owns the property.
`scheduleName`	The name of the schedule.
`stateDefinitionName`	The name of the state definition.
`stepName`	The name of the step.
`transitionDefinitionName`	The name of the state definition.
`value`	The value of the actual parameter, if creating or modifying.
`workflowDefinitionName`	The name of the workflow definition.

application

Creates or modifies an ElectricFlow object that you model and deploy to build, test , deploy, and release your software.

Required Arguments

Name	Description
`applicationName`	The name of the application.
`projectName`	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	The new name for an existing object that is being renamed.

DSL Methods for ElectricFlow Objects That Can be Nested Inside

- applicationTier
- component
- emailNotifier
- environmentTemplateTierMap
- process
- property
- snapshot
- tierMap

applicationTier

Creates or modifies a logical grouping of a components that are part of an application and the resources on which they will be deployed.

When `applicationTier` has a nested `component`, DSL automatically calls the `addComponentToApplicationTier` API.

Required Arguments

Name	Description
`applicationName`	The name of the application.
`applicationTierName`	Name of the tier that must be unique within the application.
`projectName`	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	New name for an existing object.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- component
- property

artifact

Creates or modifies a top-level object containing various metadata about artifacts, including artifact versions, a name template for published artifact versions, artifact specific properties, and access control entries to specify privileges.

Required Arguments

None.

Optional Arguments

Name	Description
`artifactKey`	The `artifactKey` component of the GroupId/ArtifactKey/Version (GAV) coordinates.
`artifactName`	The name of the artifact.
`artifactVersionNameTemplate`	The artifactVersion name template.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`groupId`	The groupId component of the GroupId/ArtifactKey/Version (GAV) coordinates.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- artifactVersion
- property

artifactVersion

Creates or modifies a collection of various metadata, returning `0` to `N` files that were published to an artifact repository.

Required Arguments

None.

Optional Arguments

Name	Description
`artifactKey`	The `artifactKey` component of the GroupId/ArtifactKey/Version (GAV) coordinates.
`artifactName`	The name of the artifact containing the `artifactVersion`.
`artifactVersionName`	The name of the `artifactVersion`.
`artifactVersionState`	The state of the `artifactVersion`.
`dependentArtifactVersions`	The set of `artifactVersion`s on which this `artifactVersion` depends. An alternate name for this argument is `dependentArtifactVersion`.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`groupId`	The groupId component of the GAV (GroupId/ArtifactVersionId/Version) coordinates.
`jobStepId`	The id of the job step; used to make a project association.
`newName`	New name for an existing object that is being renamed.
`removeAllDependentArtifactVersions`	If this is set to `true` or `1`, all dependencies will be removed.
`repositoryName`	The name of the artifact repository.
`version`	The version component of the GAV (GroupId/ArtifactVersionId/Version) coordinates.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

component

Creates or modifies the object based on a specific version of an artifact and defined in an application.

Required Arguments

Name	Description
`componentName`	The name of the component
`projectName`	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`actualParameters`	Parameters passed as arguments to the application component. The alternate argument name is `actualParameter`.
`applicationName`	The name of an application to which the component is scoped.
`clearActualParameters`	If this is set to `true` or `1`, the step removes all actual parameters.
`credentialName`	The name of a credential to attach to this component.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	The new name for an existing object.
`pluginKey`	The key of the plugin.
`pluginName`	The name of the plugin.
`pluginParameters`	List of plugin parameters. An alternate name is `pluginParameter`.
`sourceApplicationName`	The name of source application.
`sourceComponentName`	The name of new component.
`sourceProjectName`	The name of source project.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- formalParameter
- process
- property

credential

Creates or modifies a username/password stored in the server as encrypted data. It is typically used in an agent context to authenticate with a third-party system. The password value is not available in the web context for security reasons, so this object has no `getPassword` method.

Required Arguments

Name	Description
`credentialName`	The name of the credential.
`projectName`	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	New name for an existing object.
`password`	The password for the credential. It can also be a certificate or a chunk of data.
`passwordRecoveryAllowed`	If set to `true` or `1`, ElectricFlow recovers the password by invoking `getFullCredential` from a job step.
`userName`	The username for the credential.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

deployerApplication

Creates or modifies an application that will be deployed as part of a Deployer in a Release.

Required Arguments

Name	Description
`applicationName`	The name of the application.
`projectName`	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`applicationProjectName`	The name of the project containing specified application. If it is not specified, the default is the Release project name.
`orderIndex`	The order in which the applications are deployed, starting from 1).
`processName`	The name of the application process.
`releaseName`	The name of the Release.
`smartDeploy`	Smart deploy flag used during the `runProcess` method for the application.
`snapshotName`	The name of the snapshot.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- deployerApplication

deployerConfiguration

Creates or modifies a configuration for an application to be deployed in a release.

Required Arguments

Name	Description
`applicationName`	The name of the application.
`projectName`	The name for the project that must be unique among all projects.
`stageName`	The name of the stage of a pipeline attached to a Release if a Release is specified.

Optional Arguments

Name	Description
actualParameters	Parameters passed as arguments to the application component. The alternate argument name is actualParameter.
applicationProjectName	The name of the project containing specified application. If it is not specified, the default is the Release project name.
clearActualParameters	If this is set to true or 1, the step removes all actual parameters.
environmentName	The name of the environment.
environmentProjectName	The name of the project containing the specified environment or environment template. If this is not specified, the default is Release project name.
environmentTemplateName	The name of the environment template.
environmentTemplateProjectName	The name of the project containing specified environment template. If this is not specified, the default is the environment project name.
environmentTierName	The name of the environment tier container of the property sheet that owns the property.
releaseName	The name of the Release.
skipDeploy	If set to true or 1, the application will not be deployed to the environment.

directoryProvider

Creates or modifies the configuration used to communicate with an external directory service (LDAP or ActiveDirectory).

Required Arguments

Name	Description
providerName	Name for a LDAP directory provider that must be unique.

Optional Arguments

Name	Description
`commonGroupNameAttribute`	The attribute in a group record that contains the common name of the group. If specified, it is only used when searching for groups from an external provider. It is usually used when the group name attribute is set to `distinguishedName`, because this field is not searchable.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`domainName`	The domain from which the Active Directory servers are automatically discovered.
`emailAttribute`	The attribute in a LDAP user record that contains the user's email.
`enableGroups`	Determines whether or not to enable external groups for the directory provider.
`fullUserNameAttribute`	The attribute in a user record that contains the user's full name (first and last).
`groupBase`	String prepended to the base distinguished name (DN) to construct the DN of the directory that contains group records.
`groupMemberAttributes`	Comma separated list of attribute names that can identify a member of a group.
`groupMemberFilter`	LDAP query string for the groups directory to find groups that contain a given user as a member.
`groupNameAttribute`	The attribute in a group record that contains the name of the group.
`groupSearchFilter`	LDAP query string used in group directory to enumerate group records.
`managerDn`	The name of a user who has read-only access to the LDAP or Active Directory server. It is usually a distinguished name (DN). A simple name may be used when the Active Directory server URL is being auto-discovered via DNS.
`managerPassword`	Secret value used to identify the account for the query user.
`newName`	New name for an existing object that is being renamed.
`providerType`	Type string for a directory provider: **ldap** or **activedirectory**.

Name	Description
`realm`	The realm of the LDAP directory provider. This is used to create unique usernames when there are multiple providers.
`url`	The URL of the LDAP Directory Provider server.
`useSSL`	If this argument is set to `true` or `1`, SSL is used for communication.
`userBase`	Used to construct the distinguished name (DN) of the directory that contain user records.
`userNameAttribute`	The attribute in a user record that contains the user's account name.
`userSearchFilter`	RFC 2254 LDAP query to search for a user by name.
`userSearchSubtree`	If this argument is set to `true` or `1`, ElectricFlow recursively searches the subtree below the user base.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

emailConfig

Specified information about the mail server configuration necessary for the ElectricFlow server to send an email message.

Required Arguments

Name	Description
`configName`	The email configuration name.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`mailFrom`	The email address used as the email sender address for ElectricFlow notifications.

Name	Description
`mailHost`	Name of the email server host.
`mailPort`	The port number for the email service on the server.
`mailProtocol`	Name of the email transport protocol. Supported protocol names are **SMTP** and **SMTPS**.
`mailUser`	Name of the email user on behalf of which ElectricFlow sends email notifications.
`mailUserPassword`	Password of the email user on behalf of which ElectricFlow sends email notifications.
`newName`	New name for an existing object that is being renamed.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

emailNotifier

Specifies information about the email notification that will be sent when a particular event occurs.

Required Arguments

Name	Description
`notifierName`	The name of the email notifier.

Optional Arguments

Name	Description
`applicationName`	The name of the application that is related to the target email container, a process or process step.
`componentName`	The name of the component which is related to the target email container, a process or process step.
`condition`	A fixed text or text embedding property references that is 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.

Name	Description
configName	The name for an email configuration, or text that through property expansion results in such an email configuration name.
description	Comment text describing this object that is not interpreted at all by ElectricFlow.
destinations	A list of space-separated usernames, email addresses, or email aliases or text that through property expansion results in such a list.
environmentNames	Name of the environments. An alternate argument name is environmentName.
eventType	An enumeration of valid event categories recognized by the email notifiers.
flowName	The name of the flow container of the email notifier.
flowStateName	The name of the flow container of the email notifier.
formattingTemplate	String containing the email formatting instructions for generating notifications.
gateType	The type of the gate.
groupNames	A list of the groups that receives the notification. An alternate argument name is groupName.
jobId	The primary key or name of the job container of the email notifier.
jobStepId	The primary key of the job-step container of the email notifier.
newName	New name for an existing object.
notificationType	The notification type that will be stored in the ec_ notificationType property.
pipelineName	The name of the pipeline container of the email notifier.
procedureName	The name of the procedure container of the email notifier.
processName	The name of the process container of the email notifier.
processStepName	The name of the process step container of the email notifier.
projectName	The name of the project container of the email notifier.
stageName	The name of the stage container of the email notifier.

Name	Description
stateDefinitionName	The name of the state definition container of the email notifier.
stateName	The name of the state container of the email notifier.
stepName	The name of the step container of the email notifier.
userNames	A list of names of the users who receives the notification. An alternate argument name is `userName`.
workflowDefinitionName	The name of the workflow definition container of the email notifier.
workflowName	The name of the workflow container of the email notifier.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

environment

Creates or modifies an environment to which an application is deployed.

Required Arguments

Name	Description
environmentName	The name of the environment.
projectName	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
applicationName	Application from which the environment is created.
applicationProjectName	The application project name.
description	Comment text describing this object that is not interpreted at all by ElectricFlow.
environmentEnabled	If this argument is set to `true` or `1`, the environment is enabled.
newName	New name for an object.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- environmentTier
- property

environmentTemplate

Creates or modifies a template defining an environment that can be spun up when the application is deployed.

Required Arguments

Name	Description
`environmentTemplateName`	The name of the environment template.
`projectName`	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	New name for an object that is being renamed.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property
- environmentTemplateTier

environmentTemplateTier

Creates or modifies a logical grouping of resources in an environment template.

When `environmentTemplateTier` has a nested `resource`, DSL automatically calls the `addResourceToEnvironmentTemplateTier` API.

When `environmentTemplateTier` has a nested `resourceTemplate`, DSL automatically calls the `addResourceTemplateToEnvironmentTemplateTier` API.

Required Arguments

Name	Description
`environmentTemplateName`	The name of the environment template.
`environmentTemplateTierName`	Name for the environment template tier that must be unique among all tiers for the environment template.
`projectName`	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	New name for an object.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

environmentTemplateTierMap

Creates or modifies a map containing the mapping of application tiers to the environment template tiers for an application deployment.

Required Arguments

Name	Description
`applicationName`	The name of the application.
`environmentProjectName`	The name of the environment project name.
`environmentTemplateName`	The name of the environment template.
`projectName`	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`applicationEntityRevisionId`	Revision ID of the versioned object.

Name	Description
`tierMapName`	The name of the environment template tier map. If this is not specified, ElectricFlow uses a hyphenated name consisting of the application and environment names.
`tierMappings`	The list of mappings between the application tiers and the environment template tiers. An alternate argument name is `tierMapping`.

environmentTier

Creates or modifies a logical grouping of resources in an environment.

When `environmentTier` has a nested `resource`, DSL automatically calls the `addResourceToEnvironmentTier` API.

Required Arguments

Name	Description
`environmentName`	The name of the environment.
`environmentTierName`	Name for the environment tier that must be unique among all tiers for the environment.
`projectName`	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	The new name for an object.
`resourceNames`	A list of resources to add to the environment tier. An alternate argument name is `resourceName`.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property
- resource

formalParameter

Creates or modifies an unbound parameter defined on a procedure, workflow definition, and other objects.

Required Arguments

Name	Description
`formalParameterName`	The name for this parameter that is used when the procedure is invoked to specify a value for the parameter.
`projectName`	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`applicationName`	The name of the application, if the formal parameter is on an application process.
`componentName`	The name of the component, if the formal parameter is on a component process.
`defaultValue`	If no value is provided for the parameter when the procedure is invoked, this value will be used.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`expansionDeferred`	If this argument is set to `true` or `1`, expansion for this parameter should be deferred. The parameter value will not be expanded when the procedure call is expanded, but can be expanded from a command step instead.
`flowName`	The name of the flow to which the flow state belongs.
`flowStateName`	The name of the flow state if the formal parameter is on a flow state.
`label`	The display label.
`newName`	The new name for an existing object.
`orderIndex`	The display order index, starting from 1.
`pipelineName`	The name of the pipeline, if the formal parameter is on a pipeline.
`procedureName`	The name of the procedure.

Name	Description
`processName`	The name of the process if the formal parameter is on a process.
`required`	If this argument is set to `true` or `1`, this parameter is required. The procedure will not execute unless a value is given for the parameter.
`stateDefinitionName`	The name of the state definition.
`stateName`	The name of a workflow state.
`type`	The type of a formal parameter.
`workflowDefinitionName`	The name of the workflow definition.
`workflowName`	The name of a workflow.

gateway

Creates or modifies a secure connection between two zones for sharing or transferring information between the zones.

Required Arguments

Name	Description
`gatewayName`	The gateway name.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`gatewayDisabled`	If this argument is set to `true` or `1`, the artifact repository is disabled.
`hostName1`	The domain name or IP address `resourceName2` uses to send messages to `resourceName1`.
`hostName2`	The domain name or IP address `resourceName1` uses to send messages to `resourceName2`.
`newName`	The new name for an existing object.

Name	Description
`port1`	The port number that `resourceName2` uses to send messages to `resourceName1`.
`port2`	The port number `resourceName1` uses to send messages to `resourceName2`.
`resourceName1`	The name of the first resource in a gateway specification. Other resources in this resource's zone forward messages through this resource to agents in `resourceName2's` zone.
`resourceName2`	The name of the second resource in a gateway specification. Other resources in this resource's zone forward messages through this resource to agents in `resourceName1's` zone.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

group

A group of users.

Required Arguments

Name	Description
`groupName`	Name of the group that must be unique among local groups.

Optional Arguments

Name	Description
`migrateSettings`	New group name to which settings will be migrated.
`newName`	New name for an existing object that is being renamed.
`removeAllUsers`	If this argument is set to `true` or `1`, all users are removed from this group.
`userNames`	The list of users in the group. An alternate argument name is `userName`.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

hook

Creates or modifies a resource template hook that stores a reference to a procedure in an ElectricFlow project or plugin project. When a resource template is used to create a resource pool, these procedures are invoked.

Required Arguments

Name	Description
`hookName`	Name for the hook that must be unique among all hooks.

Optional Arguments

Name	Description
`broadcast`	The broadcast flag.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`hookParameters`	The hook parameters. An alternate argument name is `hookParameter`.
`hookType`	The hook type.
`newName`	The new name for an existing object.
`procedureName`	The hook procedure name.
`procedurePluginKey`	The procedure plugin key.
`procedureProjectName`	The procedure project name.
`projectName`	The project name of the entity that owns the hook
`resourceTemplateName`	The name of the resource template.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

job

Creates or modifies an instance of a procedure run.

Required Arguments

None.

Optional Arguments

Name	Description
`destinationProject`	The project that will own the job.
`jobId`	The primary key of the job or the name of the job.
`jobNameTemplate`	The template used to determine the default name of jobs launched from a procedure.
`procedureName`	The name of the procedure that should own the job step. If this is not specified, `myStep.procedure` is used.
`projectName`	The name of the project if the `destinationProject` is not specified.
`status`	The starting status for the job.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- emailNotifier
- jobStep
- property

jobStep

Creates or modifies a step in a job.

Required Arguments

None.

Optional Arguments

Name	Description
`actualParameters`	Actual parameters passed to an invoked subprocedure. An alternate argument name is `actualParameter`.

Name	Description
`alwaysRun`	If this argument is set to `true` or `1`, this step will run even if the preceding steps fail in a way that aborts the job.
`broadcast`	If this argument is set to `true` or `1`, this step is replicated to execute (in parallel) on each of the specified resources (that is, for a pool, run the step on each of the resources in the pool).
`command`	The script to execute the functions of this step. It is passed to the step's shell for execution.
`condition`	A fixed text or text embedding property references that is 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`.
`credentialName`	The name of the credential to be used for impersonation.
`credentials`	The list of run-time credentials attached to the job step. An alternate argument name is `credential`.
`errorHandling`	Error handling method for this step.
`exclusive`	If this argument is set to `true` or `1`, the resource acquired for this step will be retained for the exclusive use of this job. This means the following: • No other job will be able to use that resource, regardless of its step limit, until this job completes. • Future steps for this job will use the resource in preference to other resources if this resource meets the needs of the steps and its step limit is not exceeded.
`exclusiveMode`	Determines the mode to use when the step acquires a resource. • If set to `none`, the default behavior for the step applies. • If set to `job`, the resource will be retained for the exclusive use of this job. • If set to `step`, the resource will be retained for the exclusive use of this step and procedure it may call. • If set to `call`, the resource will be retained for the exclusive use of all steps within the current procedure call.
`external`	If this argument is set to `true` or `1`, the step is externally managed (no state machine).
`jobStepId`	The primary key of the job step

Name	Description
`jobStepName`	The name for the new step. If this is not specified, a default name will be generated.
`logFileName`	The name of the log file for a step that is specified relative to the root directory in the job's workspace.
`parallel`	If this argument is set to `true` or `1`, this step and all adjacent steps with the flag set will run in parallel.
`parentPath`	The path to the parent job step. If this is not specified, the current job step is used.
`postProcessor`	This command runs in parallel with the main command for the step. It analyzes the log for the step and collects diagnostic information.
`precondition`	A fixed text or text embedding property references that is 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`.
`procedureName`	The name of the procedure that should own the job step. If this is not specified, `myStep.procedure` is used.
`projectName`	The name of the project for the procedure.
`releaseExclusive`	If this argument is set to `true` or `1`,the resource acquired for this step will be no longer be retained for the exclusive use of this job when this step completes.
`releaseMode`	Determines the mode to use when the step releases its resource. • If set to `none`, the default behavior applies. • If set to `release`, the resource will be available for use by any job. • If set to `releaseToJob`, the resource will be available for use by any step in this job.
`resourceName`	The name for the resource that must be unique among all resources.
`shell`	The name of the shell program that will execute the command and postprocessor for the step.
`status`	The starting status for the step.
`stepName`	The name of the procedure step that should 'own' the job step. If not specified, myStep is used.

Name	Description
`subprocedure`	The name of a procedure to invoke during this step.
`subproject`	The name of the project containing the procedure to invoke during this step.
`timeLimit`	The maximum amount of time the step can execute; abort if it exceeds this time.
`timeLimitUnits`	The units for step time limit: `seconds`, `minutes`, or `hours`.
`workingDirectory`	The working directory in which to execute the command for this step. A relative name is interpreted relative to the root directory for the job's workspace.
`workspaceName`	The name of the workspace.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- jobStep
- emailNotifier
- property

license

Creates or modifies the license data in XML format that describes the usage to which you are entitled.

Required Arguments

Name	Description
`licenseData`	Container elements for license data, which expects embedded XML as CDATA.

Optional Arguments

None.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

note

Creates or modifies notes for a Release.

Required Arguments

None.

Optional Arguments

Name	Description
`newName`	The new name for an existing object.
`note`	Object representing the notes for an entity.
`noteName`	The name of the note.
`projectName`	The project name of the entity that owns the note.
`releaseName`	The name of the Release.

`pipeline`

Creates or modifies a pipeline.

Required Arguments

Name	Description
`pipelineName`	The name of the pipeline
`projectName`	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`enabled`	If this argument is set to `true` or `1`, the pipeline is enabled.
`newName`	The new name for an existing object.
`type`	The type of pipeline

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- formalParameter
- property
- stage

plugin

Creates or modifies an add-on program used by ElectricFlow to integrate with third-party tools, custom dashboards, and unique user experiences based on roles.

Required Arguments

None.

Optional Arguments

Name	Description
`author`	The name of the plugin author.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`key`	The version independent name for the plugin.
`label`	The label to display in lists for the plugin.
`pluginName`	The name of the plugin.
`projectName`	The name of the project.
`version`	The version of the plugin.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

procedure

Creates and modifies the steps that execute some task.

Required Arguments

Name	Description
`procedureName`	The name for the procedure that must be unique within the project.
`projectName`	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`credentialName`	The name of a credential to attach to this procedure.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`jobNameTemplate`	The template used to determine the default name of jobs launched from a procedure.
`newName`	The new name for an existing object.
`resourceName`	The name of the default resource for this procedure.
`timeLimit`	The maximum amount of time the step can execute. The step is aborted if it exceeds this time.
`timeLimitUnits`	The units for step time limit: `seconds`, `minutes`, or `hours`.
`workspaceName`	The name of the default workspace for this procedure.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- emailNotifier
- formalParameter
- property
- step

process

Creates or modifies an application or component process.

Required Arguments

Name	Description
`processName`	The name of the process.
`projectName`	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`applicationName`	The name of the application if the process is owned by an application.
`componentApplicationName`	If this is specified, the component is scoped to this application, not the project.
`componentName`	The name of the component if the process is owned by a component.
`credentialName`	The name of a credential to attach to this process.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	The new name for an existing object.
`processType`	The type of action performed by the process.
`timeLimit`	The maximum amount of time the step can execute. It is aborted if it exceeds this time.
`timeLimitUnits`	The units for step time limit: `seconds`, `minutes`, or `hours`.
`workspaceName`	The name of the default workspace for this process.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- emailNotifier
- formalParameter
- processDependency
- property

processDependency

Creates or modifies a dependency between process steps.

Required Arguments

Name	Description
`processName`	The name of the process.
`processStepName`	The name of the process step.
`projectName`	The name for the project that must be unique among all projects.
`targetProcessStepName`	The name of the target process step.

Optional Arguments

Name	Description
`applicationName`	The name of the application if the process is owned by an application.
`branchCondition`	The branch condition.
`branchConditionName`	The name of the branch condition.
`branchConditionType`	The type of branch condition.
`branchType`	The type of branch.
`componentApplicationName`	If specified, the component is scoped to this application, not the project.
`componentName`	The name of the component if the process is owned by a component.

processStep

Creates or modifies a step in an application or component process.

Required Arguments

Name	Description
`processName`	The name of the process.
`processStepName`	The name of the process step.
`projectName`	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`actualParameters`	Actual parameters passed to an invoked subprocedure or process. An alternate argument name is `actualParameter`.
`afterProcessStep`	If specified, the process step will be placed after the named process step.
`applicationName`	The name of the application if the process is owned by an application.
`applicationTierName`	If the step references an application tier, this is the name of the application tier.
`beforeProcessStep`	If specified, the process step will be placed before the named process step.
`clearActualParameters`	If this argument is set to `true` or `1`, the step will remove all actual parameters.
`componentApplicationName`	If specified, the component is scoped to this application, not the project.
`componentName`	The name of the component if the process is owned by a component.
`credentialName`	The name of the credential.
`dependencyJoinType`	The join type for incoming dependencies.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`errorHandling`	Error handling method for this step.
`includeCompParameterRef`	If this argument is set to `true` or `1`, the actual parameters will be generated from the component properties This works only for artifact components.
`newName`	The new name for an existing object.
`processStepType`	The type of the process step
`subcomponent`	If the step referencing a component process, the name of the component.
`subcomponentApplicationName`	If referencing a component process, the name of the component application (if it is not project scoped).

Name	Description
`subcomponentProcess`	If referencing a component process, the name of the component process.
`subprocedure`	If referencing a procedure, the name of the procedure.
`subproject`	If referencing a procedure, the name of the procedure's project.
`timeLimit`	The maximum amount of time the step can execute. The step aborts if it exceeds this time.
`timeLimitUnits`	The units for step time limit: `seconds`, `minutes`, or `hours`.
`workspaceName`	The name of the workspace.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- emailNotifier
- property

project

Creates and modifies a group of related procedures and schedules.

Required Arguments

Name	Description
`projectName`	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`credentialName`	The name of the credential.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	The new name for an existing object.
`resourceName`	The name for the resource that must be unique among all resources.

Name	Description
`tracked`	If this argument is set to `true` or `1`, Change Tracking is enabled for this project.
`workspaceName`	The name of the workspace.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- application
- component
- credential
- deployerApplication
- environment
- environmentTemplate
- pipeline
- procedure
- process
- property
- release
- resourceTemplate
- schedule
- stage
- workflowDefinition

property

Creates or modifies a custom attribute attached to any ElectricFlow object. This may be a key, string-value pair or a complex structure, where the value is a reference to a property sheet containing nested properties.

Required Arguments

Name	Description
`propertyName`	Name for the property that must be unique within the property sheet.

Optional Arguments

Name	Description
applicationName	The name of the application container of the property sheet that owns the property.
applicationTierName	The name of the application tier container of the property sheet that owns the property.
artifactName	The name of the artifact container of the property sheet that owns the property.
artifactVersionName	The name of the artifactVersion container of the property sheet that owns the property.
componentName	The name of the component container of the property sheet that owns the property.
configName	The name of the emailConfig container that owns the property.
counter	Whether or not the property is used as a counter.
credentialName	The name of the credential container of the property sheet that owns the property.
description	Comment text describing this object that is not interpreted at all by ElectricFlow.
environmentName	The name of the environment container of the property sheet that owns the property.
environmentTemplateName	The name of the environment template container of the property sheet that owns the property.
environmentTemplateTierName	The name of the environment template tier container of the property sheet that owns the property.
environmentTierName	The name of the environment tier container of the property sheet that owns the property.
expandable	Whether the property is recursively expandable.
extendedContextSearch	For simple property names, whether to search objects in the hierarchy to find the specified property.
flowName	The name of the flow container of the property sheet that owns the property.
flowRuntimeName	The name of the flow runtime container of the property sheet that owns the property.

Name	Description
`flowRuntimeStateName`	The name of the flow state container of the property sheet that owns the property.
`flowStateName`	The name of the flow state container of the property sheet that owns the property.
`flowTransitionName`	The name of the flow transition container of the property sheet that 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.
`jobId`	The primary key or name of the job container of the property sheet that owns the property.
`jobStepId`	The primary key of the job-step container of the property sheet that owns the property.
`newName`	New name for an existing object.
`notifierName`	The name of the notifier container of the property sheet that owns the property.
`objectId`	The object id as returned by `findObjects`.
`path`	The property path string.
`pipelineName`	The name of the pipeline container of the property sheet that owns the property.
`pluginName`	The name of the plugin container of the property sheet that owns the property.
`procedureName`	The name of the procedure container of the property sheet that owns the property.
`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 that owns the property.
`propertySheetId`	The primary key of the property sheet that owns the property.
`propertyType`	The type of property.

Name	Description
`releaseName`	The name of the release container of the property sheet that owns the property.
`repositoryName`	The name of the repository container of the property sheet that owns the property.
`resourceName`	The name of the resource container of the property sheet that owns the property.
`resourcePoolName`	The name of the resource pool container of the property sheet that owns the property.
`resourceTemplateName`	The name of the resource template container of the property sheet that owns the property.
`scheduleName`	The name of the schedule container of the property sheet.
`snapshotName`	The name of the snapshot container of the property sheet that owns the property.
`stageName`	The name of the stage container of the property sheet that owns the property.
`stateDefinitionName`	The name of the state definition container of the property sheet that owns the property.
`stateName`	The name of the state container of the property sheet that owns the property.
`stepName`	The name of the step container of the property sheet that owns the property.
`systemObjectName`	The system object.
`taskName`	The name of the task that owns property sheet.
`transitionDefinitionName`	The name of the transition definition container of the property sheet that owns the property.
`transitionName`	The name of the transition container of the property sheet which owns the property.
`userName`	The name of the user container of the property sheet that owns the property.
`value`	The value of the property.
`workflowDefinitionName`	The name of the workflow definition container of the property sheet that owns the property.

Name	Description
`workflowName`	The name of the workflow container of the property sheet that owns the property.
`workspaceName`	The name of the workspace container of the property sheet.
`zoneName`	The name of the zone container of the property sheet.

release

Creates and modifies a Release.

Required Arguments

Name	Description
`projectName`	The name for the project that must be unique among all projects.
`releaseName`	The name of the Release.

Optional Arguments

Name	Description
`actualParameters`	Parameters passed as arguments to the application. The alternate argument name is `actualParameter`.
`clearActualParameters`	If this is set to `true` or `1`, the task removes all actual parameters.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	The name of the pipeline.
`pipelineName`	Whether or not to disable the repository.
`pipelineProjectName`	The name of the project containing the specified pipeline. If this is not specified, the default is the Release project name.
`plannedEndDate`	The date when this release is expected to end (for example, 2006-05-15).
`plannedStartDate`	The date when this release is expected to begin (for example, 2006-05-15).

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- note
- property

repository

Creates or modifies the object that stores artifact versions. It primarily contains information about how to connect to a particular artifact repository.

Required Arguments

Name	Description
`repositoryName`	The name of the repository.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	New name for an existing object.
`repositoryDisabled`	Whether to disable the repository.
`url`	The URL for contacting the repository.
`zoneName`	The zone name.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

resource

Creates and modifies the agent machine that is configured to communicate with ElectricFlow and where job steps can be executed.

Required Arguments

Name	Description
`resourceName`	Name for the resource that must be unique among all resources.

Optional Arguments

Name	Description
`artifactCacheDirectory`	The artifact cache directory for this resource.
`block`	If this is set to `true` or `1`, ElectricFlow blocks on the agent ping before returning.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`hostName`	The domain name or IP address of the server machine corresponding to this resource.
`hostType`	The type of the host.
`newName`	The new name for an existing object.
`pools`	A list of arbitrary names separated by spaces, indicating the pools with which this resource is associated.
`port`	The port number used by this resource to connect to the agent. The default is the server default.
`proxyCustomization`	The proxy specific customization data. The default is `none`.
`proxyHostName`	The domain name or IP address of the proxy agent machine corresponding to this resource.
`proxyPort`	The port number used by this resource to connect to the proxy agent. The default is the server default.
`proxyProtocol`	The protocol to use when proxying to this resource. The default is `none`.
`repositoryNames`	A newline delimited list of repositories from which to retrieve artifacts.
`resourceDisabled`	If this is set to `true` or `1`, this resource will not be allocated to job steps, regardless of its step limit.
`shell`	The name of the shell program that will execute the command and postprocessor for the step.
`stepLimit`	The maximum number of steps that may execute simultaneously using this resource.

Name	Description
`trusted`	If this is set to `true` or `1`, the agent can speak to all other trusted agents in its zone. An untrusted agent can only speak to gateway agents.
`useSSL`	If this is set to `true` or `1`, SSL is used for communication.
`workspaceName`	The name of the workspace.
`zoneName`	The name for the zone that must be unique among all zones.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

resourcePool

Creates or modifies a collection of resources with a ordering policy.

Required Arguments

Name	Description
`resourcePoolName`	The name for the resource pool that must be unique among all resource pools.

Optional Arguments

Name	Description
`autoDelete`	If this is set to `true` or `1`, the pool is deleted when the last resource is deleted.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	The new name for an existing object that is being renamed.
`orderingFilter`	The JavaScript fragment that returns custom ordering of resources in a pool.
`resourceNames`	The list of resources to add or remove from the pool. An alternate argument name is `resourceName`.
`resourcePoolDisabled`	If this is set to `true` or `1`, the resource pool will not be allocated to job steps.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

resourceTemplate

Creates or modifies a template with the required information to provision and later spin up cloud resources on an on-demand basis.

Required Arguments

Name	Description
`projectName`	Name for the project that must be unique among all projects.
`resourceTemplateName`	Name for the resource template that must be unique among all resource templates.

Optional Arguments

Name	Description
`cfgMgrParameters`	Configuration manager plugin parameters. An alternate argument name is `cfgMgrParameter`.
`cfgMgrPluginKey`	Configuration manager plugin key.
`cfgMgrProcedure`	Configuration manager plugin method name.
`cfgMgrProjectName`	Configuration manager plugin project name.
`cloudProviderParameters`	Cloud provider plugin parameters. An alternate argument name is `cloudProviderParameter`.
`cloudProviderPluginKey`	Cloud provider plugin key.
`cloudProviderProcedure`	Cloud provider plugin method name.
`cloudProviderProjectName`	Cloud provider plugin project name.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	New name for an existing object.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- hook
- property

schedule

Creates or modifies the object is responsible for launching a procedure at some time in the future, possibly on a regular interval.

Required Arguments

Name	Description
`projectName`	Name for the project that must be unique among all projects.
`scheduleName`	Name for the schedule that must be unique among all schedules for the project.

Optional Arguments

Name	Description
`actualParameters`	Parameters passed to the invoked procedure, process, and/or workflow. An alternate argument name is `actualParameter`.
`applicationName`	The name of the application that owns the process.
`applicationProjectName`	The name for the project to which the application belongs.
`beginDate`	The date when this schedule will begin (for example, 2006-05-15).
`clearActualParameters`	Whether to clear actual parameters for this object.
`credentialName`	The name of the credential to use for impersonation.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`endDate`	The date when this schedule will end (for example, 2006-05-15). The end date is not included in the range of dates.
`environmentName`	The name of the environment used to determine where to run the process.
`environmentProjectName`	The name for the project to which the environment or environment template belongs.

Name	Description
`environmentTemplateName`	The name of the environment template used to determine the environment where to run the process.
`environmentTemplateProjectName`	The name for the project to which the environment template belongs.
`environmentTemplateTierMapName`	The name of the environment template tier map used to determine how to spin up a dynamic environment.
`interval`	If specified, the procedure, process, and/or workflow will be rescheduled over and over again at intervals of this length. `\"Continuous\"` means reschedule the procedure, process, and/or workflow as soon as the previous job finishes.
`intervalUnits`	The units for the rescheduling interval.
`misfirePolicy`	The misfire policy for a schedule.
`monthDays`	A list of numbers from 1 to 31 separated by spaces, indicating zero or more days of the month.
`newName`	The new name for an existing object.
`priority`	The priority of the job.
`procedureName`	The name of the procedure to invoke.
`processName`	The name of the application process to invoke.
`scheduleDisabled`	If this box is \"checked\", the schedule will run. You can disable this schedule whenever necessary.
`snapshotName`	The name of the snapshot to be used to invoke the application process.
`startTime`	The time of day to begin invoking this schedule's procedure, process, and/or workflow using a 24-hour clock (for example, 17:00).
`startingStateName`	The name of the starting state of the workflow.
`stopTime`	The time of day to stop invoking this schedule's procedure, process, and/or workflow (do not start a new job after this time) using a 24-hour clock (for example, 17:00).
`tierMapName`	The name of the tier map.
`tierResourceCounts`	The resource count per resource template tier. An alternate argument name is `tierResourceCount`.

Name	Description
`timeZone`	The time zone to use when interpreting times.
`weekDays`	The days of the week, specified by any number of names such as Monday or Tuesday, separated by spaces
`workflowName`	The name of the workflow to invoke.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

snapshot

Creates or modifies a version of an application with specific artifact versions and the state of the application at any point in time.

Required Arguments

Name	Description
`applicationName`	The name of the application.
`projectName`	The name for the project that must be unique among all projects.
`snapshotName`	The name of the snapshot that must be unique within the application.

Optional Arguments

Name	Description
`componentVersions`	The Component names and version used for snapshot. Use keyword `LATEST` to indicate latest version. The alternate argument name is `componentVersion`.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`environmentName`	The name of environment from which the snapshot will be created.
`environmentProjectName`	The name for the project to which the environment or environment template belongs.
`newName`	The new name for an existing object.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

stage

Creates or modifies a logical grouping of pipeline tasks

Required Arguments

Name	Description
`projectName`	The name of the project that must be unique among all projects.
`stageName`	The name of the stage.

Optional Arguments

Name	Description
`afterStage`	If specified, the stage will be placed after the named stage.
`beforeStage`	If specified, the stage will be placed before the named stage.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	The new name for an existing object.
`pipelineName`	The name of the pipeline.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property
- task

stateDefinition

A state definition in a workflow definition. Each workflow can contain one or more states.

Required Arguments

Name	Description
`projectName`	The name for the project that must be unique among all projects.
`stateDefinitionName`	The name used for the state definition
`workflowDefinitionName`	The name of the workflow definition.

Optional Arguments

Name	Description
`actualParameters`	The actual parameters to the state definition's process. An alternate argument name is `actualParameter`.
`clearActualParameters`	If this is set to `true` or `1`, the state definition removes all actual parameters for the process.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	The new name for an existing object.
`startable`	If this is set to `true` or `1`, the workflow can begin in this state.
`subprocedure`	The name of the sub procedure.
`subproject`	The name of the project that contains the subprocedure.
`substartingState`	The name of the starting state to use in the `subworkflowDefinition`.
`subworkflowDefinition`	The name of the `subworkflowDefinition`.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- emailNotifier
- formalParameter
- property
- transitionDefinition

step

Creates or modifies a unit of logic that will execute on an agent.

Required Arguments

Name	Description
`procedureName`	The name of the procedure that must be unique within the project.
`projectName`	The name of the project that must be unique among all projects.
`stepName`	The name of the step that must be unique within the procedure.

Optional Arguments

Name	Description
`actualParameters`	The actual parameters passed to an invoked subprocedure. An alternate argument name is `actualParameter`.
`afterProcedureStep`	If specified, the procedure step will be placed after the named procedure step.
`alwaysRun`	If this is set to `true` or `1`, this step will run even if preceding steps fail in a way that aborts the job.
`beforeProcedureStep`	If specified, the procedure step will be placed before the named procedure step.
`broadcast`	If this is set to `true` or `1`,this step is replicated to execute (in parallel) on each of the specified resources (that is, for a pool, run the step on each of the resources in the pool).
`clearActualParameters`	If this is set to `true` or `1`, step removes all actual parameters.
`command`	A script to execute the functions of this step that is passed to the step's shell for execution.
`condition`	A fixed text or text embedding property references that is 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`.
`credentialName`	The name of the credential object.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`errorHandling`	Specifies error handling for this step.

Name	Description
`exclusive`	If this is set to `true` or `1`,the resource acquired for this step will be retained for the exclusive use of this job. This means the following: • No other job will be able to use that resource, regardless of its step limit, until this job completes. • Future steps for this job will use the resource in preference to other resources, if this resource meets the needs of the steps and its step limit is not exceeded.
`exclusiveMode`	Determines the mode to use when the step acquires a resource. If set to 'none', then the default behavior for the step applies. If set to 'job', then the resource will be retained for the exclusive use of this job. If set to 'step', then the resource will be retained for the exclusive use of this step and procedure it may call. If set to 'call', then the resource will be retained for the exclusive use of all steps within the current procedure call.
`logFileName`	The name of the log file for a step; specified relative to the root directory in the job's workspace.
`newName`	The new name for an existing object.
`parallel`	If this is set to `true` or `1`, this step and all adjacent steps with the flag set will run in parallel.
`postProcessor`	This command runs in parallel with the main command for the step; it analyzes the log for the step and collects diagnostic information.
`precondition`	A fixed text or text embedding property references that is 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.
`releaseExclusive`	If this is set to `true` or `1`, the resource acquired for this step will be no longer be retained for the exclusive use of this job when this step completes.
`releaseMode`	Determines the mode to use when the step releases its resource. If set to 'none', the default behavior applies. If set to 'release', then the resource will be available for use by any job. If set to 'releaseToJob', then the resource will be available for use by any step in this job.
`resourceName`	Name for the resource that must be unique among all resources.
`shell`	Name of the shell program that will execute the command and postprocessor for the step.
`subprocedure`	Name of a procedure to invoke during this step.

Name	Description
`subproject`	Name of the project containing the procedure to invoke during this step.
`timeLimit`	Maximum amount of time the step can execute; abort if it exceeds this time.
`timeLimitUnits`	Units for step time limit: seconds, minutes, or hours.
`workingDirectory`	Working directory in which to execute the command for this step. A relative name is interpreted relative to the root directory for the job's workspace.
`workspaceName`	The name of the workspace.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- emailNotifier
- property

task

A representation of task within a stage or gate.

Required Arguments

Name	Description
`projectName`	The name for the project that must be unique among all projects.
`taskName`	The name of the task

Optional Arguments

Name	Description
`actualParameters`	Actual parameters passed to an invoked subprocedure. An alternate argument name is `actualParameter`.
`advancedMode`	If this is set to `true` or `1`, advanced mode is enabled so that a template can be created from a snapshot.
`afterTask`	If specified, the task will be placed after the named task.

Name	Description
`approvers`	A list of task approvers who receive the notification. An alternate argument name is `approver`.
`beforeTask`	If specified, the task will be placed before the named task.
`clearActualParameters`	If this is set to `true` or `1`, the task removes all actual parameters.
`credentials`	Credentials to be used in the task. An alternate argument name is `credential`.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`enabled`	If this is set to `true` or `1`, the task is enabled.
`environmentName`	The name of the environment to create using an environment template.
`environmentTemplateName`	The name of the environment template.
`errorHandling`	Error handling method for this task.
`gateType`	The type of the gate.
`keepOnError`	If this is set to `true` or `1`, ElectricFlow keeps environment on error. The default is `false`.
`newName`	The New name for an existing object.
`notificationTemplate`	The string containing email formatting instructions for generating notifications.
`pipelineName`	The name of the pipeline
`skippable`	If this is set to `true` or `1`, a task can be skipped.
`snapshotName`	Name of the snapshot associated with the application.
`stageName`	Name of the stage to which this task belongs.
`startTime`	The time of day to begin invoking this task using a 24-hour clock { (for example, 17:00).
`subapplication`	The name of the application that owns the subprocess.
`subprocedure`	If referencing a procedure, the name of the procedure.
`subprocess`	The name of the process.

Name	Description
`subproject`	If referencing a procedure, the name of the procedure's project.
`taskProcessType`	The type of the process a task can invoke.
`taskType`	The type of the task.
`tierResourceCounts`	Resource count per resource template tier. An alternate argument name is `tierResourceCount`.
`workspaceName`	The name of the workspace.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

tierMap

A map to hold mappings between application and an environment tiers.

Required Arguments

Name	Description
`applicationName`	The name of the application
`environmentName`	The name of the environment.
`environmentProjectName`	The name of the environment's project name.
`projectName`	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
`applicationEntityRevisionId`	Revision ID of the versioned object
`tierMapName`	The name of the tier map. If not specified, ElectricFlow uses a hyphenated name consisting of the application and environment names.
`tierMappings`	The list of mappings between the application tiers and the environment tiers. An alternate name is `tierMapping`.

transitionDefinition

Defines how a workflow must transition from one state to another.

Required Arguments

Name	Description
`projectName`	Name for the project that must be unique among all projects.
`stateDefinitionName`	The name used for the state definition.
`transitionDefinitionName`	The name used for the transition definition.
`workflowDefinitionName`	The name of the workflow definition.

Optional Arguments

Name	Description
`actualParameters`	The actual parameters to the transition's target state. An alternate name is `actualParameter`.
`clearActualParameters`	If this is set to `true` or `1`, the transition should remove all actual parameters from the target state.
`condition`	A fixed text or text embedding property references that is evaluated into a logical TRUE or FALSE. If the string is empty or this is set to `false` or `0`,the text is interpreted as `FALSE`. If the string is not empty or this is set to `true` or `1`,the text is interpreted as `TRUE`.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	New name for an existing object that is being renamed.
`targetState`	Target state for the transition definition.
`trigger`	Specifies the type of trigger for this transaction.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

user

A user defines an account used to log into the system and control access to ElectricFlow objects.

Required Arguments

Name	Description
`userName`	The username.

Optional Arguments

Name	Description
`email`	Email address of the user.
`fullUserName`	Full name of the user.
`groupNames`	List of groups that this user is in. An alternate name is `groupName`.
`migrateSettings`	Migrate the user or group settings to this username.
`newName`	New name for an object that is being renamed.
`password`	The user's password.
`removeFromAllGroups`	If this is set to `true` or `1`, ElectricFlow removes this user from all groups.
`sessionPassword`	Session user's password.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

workflowDefinition

A top-level workflow object, which is a container for states, and transitions, and other information defining your workflow.

Required Arguments

Name	Description
`projectName`	Name for the project that must be unique among all projects.
`workflowDefinitionName`	The name of the workflow definition.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	New name for an object that is being renamed.
`workflowNameTemplate`	The template used to name instances of this workflow definition.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property
- stateDefinition

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.

Required Arguments

Name	Description
`workspaceName`	The workspace name.

Optional Arguments

Name	Description
`agentDrivePath`	On Windows, the path name to the root directory of a workspace, specified with a drive letter.
`agentUncPath`	On Windows, the path name to the root directory of a workspace, specified with a UNC path.

Name	Description
`agentUnixPath`	On UNIX, the path name to the root directory of a workspace.
`credentialName`	The name of the impersonation credential to attach to this workspace.
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`local`	True if the workspace is 'local'.
`newName`	New name for an existing object that is being renamed.
`workspaceDisabled`	True means this workspace is disabled.
`zoneName`	The zone name.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

zone

A zone or top-level network created as a way to partition a collection of agents to secure them from use by other groups.

Required Arguments

Name	Description
`zoneName`	The zone name.

Optional Arguments

Name	Description
`description`	Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName`	New name for an object that is being renamed.

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

Troubleshooting and FAQs

Troubleshooting

Issue	What to Do	Links to Related Topics

Get help on the DSL methods	To get the complete list of supported DSL methods, enter the following command:
	`ectool evalDsl --describe 1`
	
	To get help on a particular DSL method, enter the following command:
	`ectool evalDsl <dsl_method_name> --describe 1`
	Example: `ectool evalDsl procedure --describe 1`
		Getting Started with DSL on page 836

The DSL script completes successfully using the `evalDsl` command; however, the ElectricFlow objects are not created or updated as expected.	Use the `debug` option to trace and debug the script processing using the following command: `ectool evalDsl <dsl> --debug 1` The `debug` option will allow `evalDsl` to generate debug output that you can use to follow the DSL script processing by `evalDsl` and identify any possible issue with the DSL script.	Creating and Running DSL Scripts on page 837

FAQs

I am comfortable with Perl and am already using the ElectricFlow Perl API for my scripting purposes. Do I need to switch to ElectricFlow DSL?

Answer: ElectricFlow DSL being a dynamic scripting language provides a cleaner and much easier syntax for non-technical users to understand. However, ElectricFlow Perl API and the RESTful API are both supported as well, and you can continue to use them both if they suit your scripting needs.

ElectricFlow DSL is based on Groovy. So, are all Groovy constructs available for use in a DSL script?

Answer: Yes, most Groovy constructs such as closures, named arguments, and so on can be used in your DSL script.

Is there a way to create a DSL script for the ElectricFlow objects that I have created through the UI or using the ElectricFlow Perl API?

Answer: Yes, you can use the `generateDsl` command to create a DSL script for any ElectricFlow object.

Command: `ectool generateDsl [path]`

Example: `ectool generateDsl /projects/Default/applications/MyApp`

How to create or update access control for an object

You should use one of the following patterns for defining access control for objects in your DSL script. These patterns allow the DSL runtime engine to deterministically find the correct ElectricFlow object for creating or updating access control entries (ACEs).

- Access control entries can be nested inside the DSL method of the object for which the access control is being defined. For example, the following script provides the user named "*joe*" full access to project called *Foo*.

  ```
  project 'Foo', {
            aclEntry principalName: 'joe',
              principalType: 'user',
              readPrivilege: 'allow',
              modifyPrivilege: 'allow',
              changePermissionsPrivilege: 'allow'
  }
  ```

- Use the `objectType` argument to declare the type of object for which the access control is being defined if the access control method is not enclosed within the DSL method for the object. This can be used to define ACEs for objects such as server and other system objects because they do not have a DSL method similar to other regular objects.

  ```
  // Removing any previously set ACL for 'Everyone'
   deleteAclEntry (
       principalType: 'group',
       principalName: 'Everyone',
       objectType : 'server',
       systemObjectName : 'server'
   )

   aclEntry principalName: 'joe',
       principalType: 'user',
       objectType : 'server',
       systemObjectName : 'server',
       readPrivilege: 'allow',
       modifyPrivilege: 'allow',
       changePermissionsPrivilege: 'allow'
  ```

  or

  ```
  aclEntry principalName: 'joe',
      principalType: 'user'
      objectType : 'systemObject',
      systemObjectName : 'resources',
      readPrivilege: 'allow',
      modifyPrivilege: 'allow',
      changePermissionsPrivilege: 'allow'
  ```

Using Groovy and JRuby

When ElectricFlow is installed on Windows or UNIX (using the agent or tools installation), copies of Groovy (ec-groovy) and JRuby (ec-jruby) are installed.The installation package includes Groovy 2.4.3 and JRuby 1.7.18.

The default UNIX directories are:

- `/opt/electriccloud/electriccommander/bin/ec-jruby`
- `/opt/electriccloud/electriccommander/bin/ec-groovy`

The default Windows directories are:

- `C:\Program Files\Electric Cloud\ElectricCommander\bin\ec-groovy`
- `C:\Program Files\Electric Cloud\ElectricCommander\bin\ec-jruby`

ElectricFlow does not automatically add these to your path because:

- We do not want the ElectricFlow installation to interfere with existing scripts you may run, which are dependent on finding another copy of Groovy or JRuby you already use.
- Some special environment variables need to be set before calling Groovy or JRuby.

Both of these issues are addressed with small wrapper programs called *ec-groovy* and *ec-jruby*. They are installed as part of ElectricFlow, and are in directories added to your path. When ec-groovy or ec-jruby runs, it sets the environment variables, finds the ElectricFlow copy of Groovy or JRuby, and calls it, passing all of its parameters to Java.

ec-groovy

To run ec-groovy:

1. Set `COMMANDER_HOME` and `COMMANDER_DATA` as environment variables.
2. Enter `ec-groovy <yourGroovyOptions> <GroovyScriptName>.groovy` on the command line.

There is no language-specific binding for Groovy. Use the RESTful API to communicate with the ElectricFlow server.

This is an example of a Groovy script:

```
import groovyx.net.http.RESTClient

@Grab(group = 'org.codehaus.groovy.modules.http-builder', module = 'http-builder',
version = '0.7.1')

def commanderServer = 'https://' + System.getenv('COMMANDER_SERVER')
def commanderPort = System.getenv('COMMANDER_HTTPS_PORT')

def sessionId = System.getenv('COMMANDER_SESSIONID')

def client = new RESTClient(commanderServer + ":" + commanderPort)
client.ignoreSSLIssues()

def resp = client.get( path : '/rest/v1.0/projects/' ,headers:
['Cookie': "sessionId=" + sessionId, 'Accept': 'application/json'] )

println resp.getData()
```

Groovy also allows for run-time resolution of artifacts that can download artifacts across the internet. To disable this ability or to allow only trusted repositories to download (whitelist trusted repositories), write a `grapeConfig.xml` file and put it in the `$DATADIR/grape` directory.

This is an example of a `grapeConfig.xml` file without internet repositories:

```
<!--
      Licensed to the Apache Software Foundation (ASF) under one
      or more contributor license agreements.  See the NOTICE file
      distributed with this work for additional information
      regarding copyright ownership.  The ASF licenses this file
      to you under the Apache License, Version 2.0 (the
      "License"); you may not use this file except in compliance
      with the License.  You may obtain a copy of the License at
      http://www.apache.org/licenses/LICENSE-2.0
      Unless required by applicable law or agreed to in writing,
      software distributed under the License is distributed on an
      "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY
      KIND, either express or implied.  See the License for the
      specific language governing permissions and limitations
      under the License.
-->
   <ivysettings>
      <settings defaultResolver="downloadGrapes"/>
      <resolvers>
         <chain name="downloadGrapes" returnFirst="true">
            <filesystem name="cachedGrapes">
               <ivy pattern="${user.home}/.groovy/grapes/[organisation]/
               [module]/ivy-[revision].xml"/>
               <artifact pattern="${user.home}/.groovy/grapes/[organisation]/[module]/
               [type]s/[artifact]-[revision](-[classifier]).[ext]"/>
            </filesystem>
            <ibiblio name="localm2" root="file:${user.home}/.m2/repository/"
            checkmodified="true" changingPattern=".*" changingMatcher="regexp" m2compa
            tible="true"/>
         </chain>
      </resolvers>
   </ivysettings>
```

ec-jruby

To run ec-jruby:

1. Set `COMMANDER_HOME` and `COMMANDER_DATA` as environment variables.
2. Enter `ec-jruby <yourJRubyOptions> <JRubyScriptName>.rb` on the command line.

There is no language-specific binding for JRuby. Use the RESTful API to communicate with the ElectricFlow server.

Example of a JRuby script:

```
require 'net/https'
require 'cgi'

uri = URI.parse("https://" + ENV["COMMANDER_SERVER"] + ":" +
ENV["COMMANDER_HTTPS_PORT"] + "/rest/v1.0/projects")

http = Net::HTTP.new(uri.host, uri.port)
http.use_ssl = true
http.verify_mode = OpenSSL::SSL::VERIFY_NONE

request = Net::HTTP::Get.new(uri.request_uri)

cookie = CGI::Cookie.new('sessionId',ENV["COMMANDER_SESSIONID"])
request['Cookie'] = cookie.to_s

response = http.request(request)

puts response.body
```

Glossary

This glossary is a reference topic containing short descriptions for ElectricFlow objects, terms, and concepts.

Term	Description
access control	An access control list (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 ElectricFlow 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 ElectricFlow 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 ElectricFlow 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 ElectricFlow 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 ElectricFlow component that runs on each machine where job steps can execute. The agent works under the ElectricFlow 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 the **Web Interface Help** > **Resources** topic for more information.
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 the **Artifact Management** > **Artifact objects** > **Repository** topic for more information.
artifact version	An artifact version is a collection of `0` to `N` files that were published to an artifact repository.
backing store	The backing store is the directory on the repository server where artifact versions are stored. By default, the backing store is the <datadir>/repository-data directory in the repository installation—this default setting can be changed.
child step	A nested step that appears as a substep in the Job Details page when a step is either of the following: • A subprocedure step • A step that dynamically creates subordinate ("child") steps using the `createJobStep` API command For details about the createJobStep API command, see the ElectricFlow API Guide at http://docs.electric-cloud.com/eflow_doc/FlowIndex.html).
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 ElectricFlow 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.

Term	Description
description	A description is an optional plain text or HTML description for an object. Description text is for your use, ElectricFlow does not use this information. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>`
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 ElectricFlow *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 ElectricFlow. When the ec-perl wrapper runs, it sets up the environment, finds, and calls ElectricFlow's copy of Perl, passing all of its parameters to Perl.
ectool	*ectool* is the ElectricFlow command-line application that provides control over the ElectricFlow system if you prefer using a command-line interface rather than the ElectricFlow web interface. Most functions that can be invoked through the ElectricFlow 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 ElectricFlow engine for continuous integration, integrating with numerous Source Control Management (SCM) systems. ElectricSentry is installed automatically with ElectricFlow and is contained in a 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 ElectricFlow server and your mail server.
email notifier	After setting up the ElectricFlow 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.

Term	Description
Event log	See log.
Everyone	A special intrinsic access control on page 918group that includes **all** users.
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 are 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 ElectricFlow 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 ElectricFlow server. See the **ElectricFlow Help** > **Web Interface Help** > **Users and Groups** > **Groups** topic for more information.
impersonation	Impersonation is a mechanism that allows a job step to execute under a particular login account (the ElectricFlow agent "impersonates" a particular user during the execution of that step). Impersonation is implemented using credentials. See the *ElectricFlow User Guide* for more information.
inheritance	A feature of the ElectricFlow 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.

Term	Description
intrinsic property	Intrinsic properties represent attributes that describe the object to which they are attached. ElectricFlow automatically provides intrinsic properties for each similar type object within ElectricFlow. 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 ElectricFlow procedure. A new job is created each time you **run** (execute) a procedure.
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 ElectricFlow 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 pop-up 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 enter:
	`$[projectName]_$[/increment /myproject/jobCounter]_$[timestamp]`
	which produces a name like:
	`projectFoo_1234_20140102130321`
	You can enter 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 ElectricFlow 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.

Term	Description
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 ElectricFlow for a particular job. Each job workspace is allocated as the child of a workspace root directory. See workspace on page 932.
local group	A group defined *inside* ElectricFlow, 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* ElectricFlow, 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 ElectricFlow. We recommend 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 ElectricFlow only. See user.
log	ElectricFlow 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 **ElectricFlow Help** > **Web Interface Help** > **Event Log** topic for more information.

Term	Description
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 on page 928 .
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 ElectricFlow. Plugins are delivered as a JAR file containing the functional implementation. When a plugin is installed, the ElectricFlow 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. For a complete list of plugins that are bundled with ElectricFlow, see Appendix A: Plugins That are Bundled with ElectricFlow.
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, ElectricFlow can choose any available resource within that pool.
postp	*postp* is a postprocessor included with ElectricFlow. 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.

Term	Description
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. ElectricFlow 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.
privileges	ElectricFlow supports four privilege types for access control and 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 ElectricFlow 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.

Term	Description
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 ElectricFlow 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.
property	A property is a `name-value` pair associated with ElectricFlow 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*. ***ElectricFlow 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 ElectricFlow 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 ElectricFlow 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.

Term	Description
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 ElectricFlow 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.
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	ElectricFlow 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 ElectricFlow 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

Term	Description
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 *backing store*. By default, the backing store 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 ElectricFlow server.
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." ElectricFlow supports two types of resources: • **Standard** - specifies a machine running the ElectricFlow agent on one of the supported agent platforms • **Proxy** - requires SSH keys for authentication. You can create proxy resources (agents and targets) for ElectricFlow 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 ElectricFlow Home page facility and records the location of a page you visit frequently (either inside or outside of ElectricFlow), 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.

Term	Description
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.
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 ElectricFlow procedure. • Plugin step - These include task-specific steps. Depending on which step-type you choose, the information you need to enter is somewhat different. Some of the step types bundled with ElectricFlow include: ◦ Publish or retrieve artifact version ◦ Send Email ◦ Various SCM step types ◦ Build tools
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`".

Term	Description
system object	This is a special object whose access control lists are used to control access to some ElectricFlow internals. System objects are: `admin`, `artifactVersions`, `directory`, `emailConfigs`, `forceAbort`, `licensing`, `log`, `plugins`, `priority`, `projects`, `repositories`, `resources`, `server`, `session`, and `workspaces`.
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. Enter 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.

Term	Description
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 ElectricFlow objects. A user can be defined externally in an LDAP or Active Directory repository, or locally in ElectricFlow. See local user.
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 ElectricFlow 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.

Term	Description

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 ElectricFlow 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 ElectricFlow 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).