

ElectricFlow 6.3 API Guide

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

Copyright © 2002 – 2016 Electric Cloud, Inc. All rights reserved.

Published 5/12/2016

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

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

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

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

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

Trademarks

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

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

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

Contents

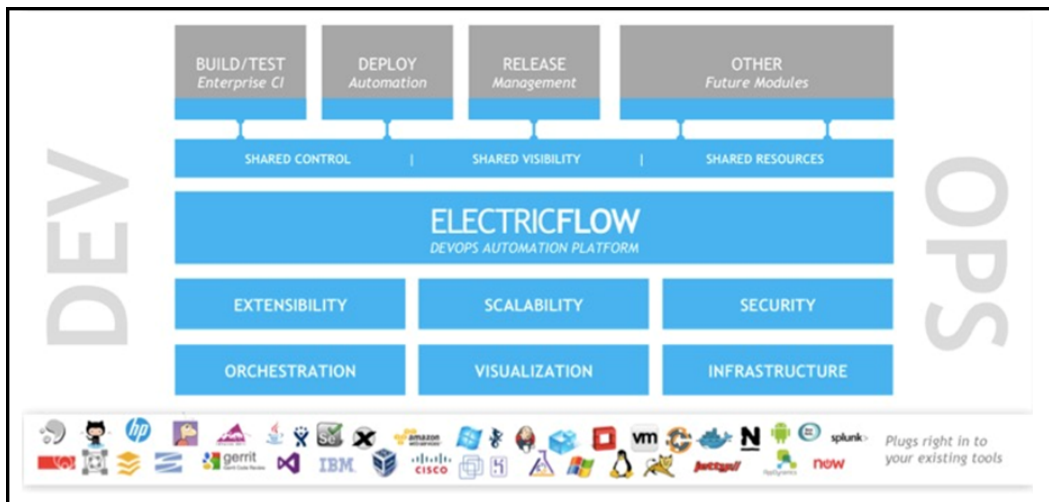
Introduction to ElectricFlow	1
Web-Based System	1
Automation Platform	6
What Makes ElectricFlow Unique?	6
ElectricFlow Architecture	6
Simple Architectural Overview	7
Expanded Remote Configuration	8
Other Configurations	9
Build-Test Automation	9
Deployment Automation	12
Pipeline Management	16
Roadmap to the ElectricFlow APIs	16
ElectricFlow Glossary	18
Using the ElectricFlow Perl API	40
Using ectool	41
Logging In	41
Global Arguments (Optional)	41
Passing Lists as Arguments	41
Using Perl (ec-perl)	42
Perl API Structure	42
Common Global Options	44
The Batch API	46
Using the Batch API	47
Installing ElectricFlow Perl Modules into Your Perl Distribution	48
Installing Perl Modules into the ElectricFlow Perl Distribution	48
Using API Commands in Javascript	50
ElectricFlow Perl API Commands	51
Perl API Commands Listed by Group	51
ACL Management (Access Control List)	51
Applications	51
Application Tiers	52
Artifact Management	52
Change History	54
Components	54
Credential Management	55
Database Configuration	55

Directory Provider Management	56
Dynamic Environments	56
Email Configuration and Management	58
Email Notifiers Management	59
Environments	59
Environment Tiers	60
Gateway and Zone Management	61
Job Management	61
Parameter Management	62
Pipelines and Releases	63
Plugin Management	66
Procedure Management	66
Process	67
Process Dependency Management	67
Process Step Management	68
Project Management	68
Property Management	68
Resource Management	69
Schedule Management	70
Server Management	70
Snapshot Management	71
Tier Map Management	71
User/Group Management	72
Workflow Definition Management	73
Workflow Management	73
Workspace Management	74
Miscellaneous	74
API Commands - ACL Management	76
API Commands - Applications	117
API Commands - Application Tier	122
API Commands - Artifact Management	128
API Commands – Change History	164
API Commands - Components	172
API Commands - Credential Management	184
API Commands - Database Configuration	195
API Commands - Directory Provider Management	199
API Commands - Dynamic Environments	213
API Commands - Email Configuration Management	256
API Commands - Email Notifier Management	262
API Commands - Environment	293
API Commands - Environment Tier	307
API Commands - Gateway and Zone Management	314
API Commands - Job Management	323
External Job APIs	346
API Commands - Parameter Management	369
API Commands - Pipelines and Releases	401
API Commands - Plugin Management	461

API Commands - Procedure Management	469
API Commands - Process	492
API Commands - Process Dependency	501
API Commands - Process Step	507
API Commands - Project Management	519
API Commands - Property Management	525
API Commands - Resource Management	578
API Commands - Schedule Management	600
API Commands - Server Management	612
API Commands - Snapshots	624
API Commands - Tier Map	631
API Commands - User and Group Management	637
API Commands - Workflow Definition Management	651
API Commands - Workflow Management	672
API Commands - Workspace Management	681
API Commands - Miscellaneous Management	689
API Response and Element Glossary	738
Element Glossary	760
Using the ElectricFlow RESTful API	788
Accessing the RESTful API	789
Using the RESTful API	789
RESTful API Examples	793
POST Operations Without Special Arguments	793
POST Operations with Special Arguments	795
Using the ElectricFlow DSL	797
Getting Started with DSL	797
Creating and Running DSL Scripts	798
Common Use Cases	798
Generating a DSL Script for an Existing ElectricFlow Object	799
Passing Parameters or Arguments to Your DSL Script	800
Passing DSL Parameters or Arguments from the Command Line	800
Evaluating Your DSL Script in a Specific Application Context	800
Using ElectricFlow APIs in DSL	801
Adding Support for Augmenting the Groovy Runtime classpath for ectool evalDsl	801
Debugging Your DSL Script	802
DSL Methods	802
Troubleshooting and FAQs	867
Troubleshooting	867
FAQs	867
Using Groovy and JRuby	869
ec-groovy	869
ec-jruby	871

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.



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.

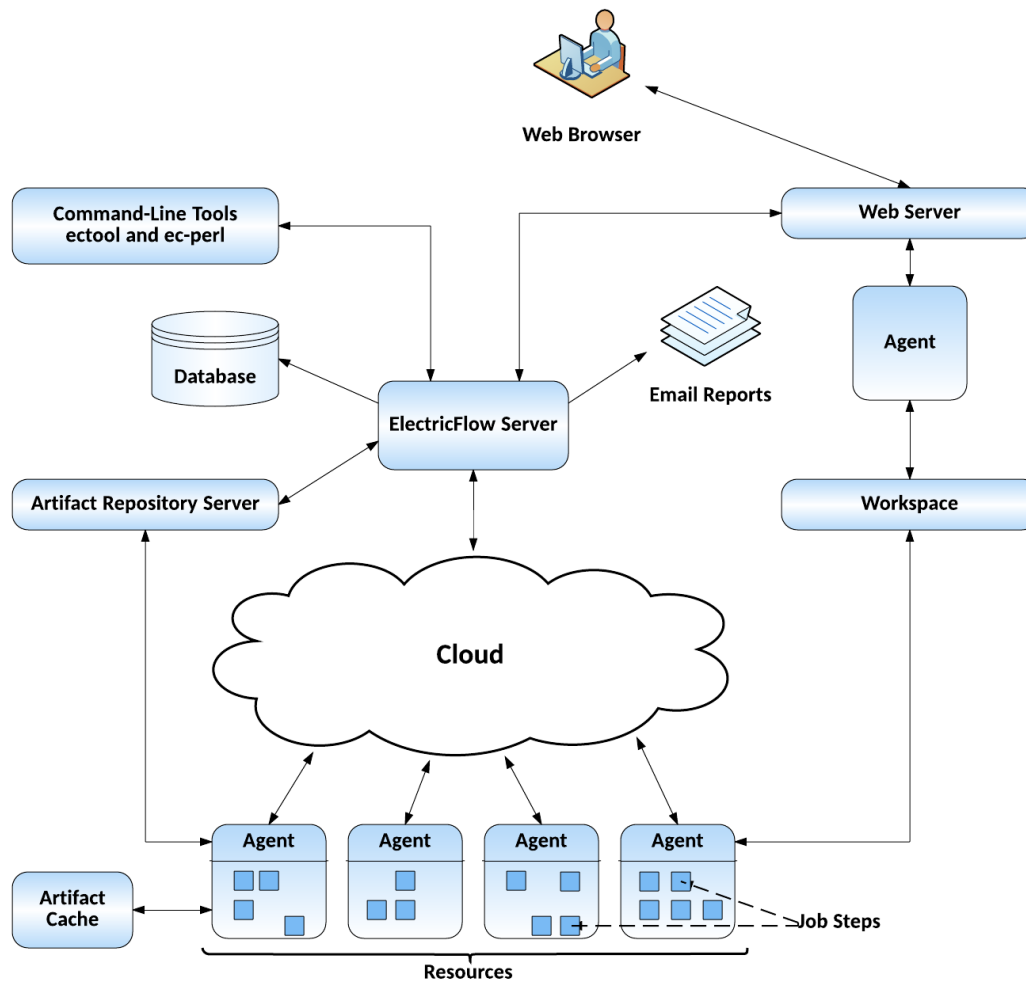
- The ElectricFlow server manages resources, issues commands, and generates reports.
- An underlying database stores commands, metadata, and log files.
- Agents execute commands, monitor status, and collect results in parallel across a cluster of servers for rapid throughput.

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.



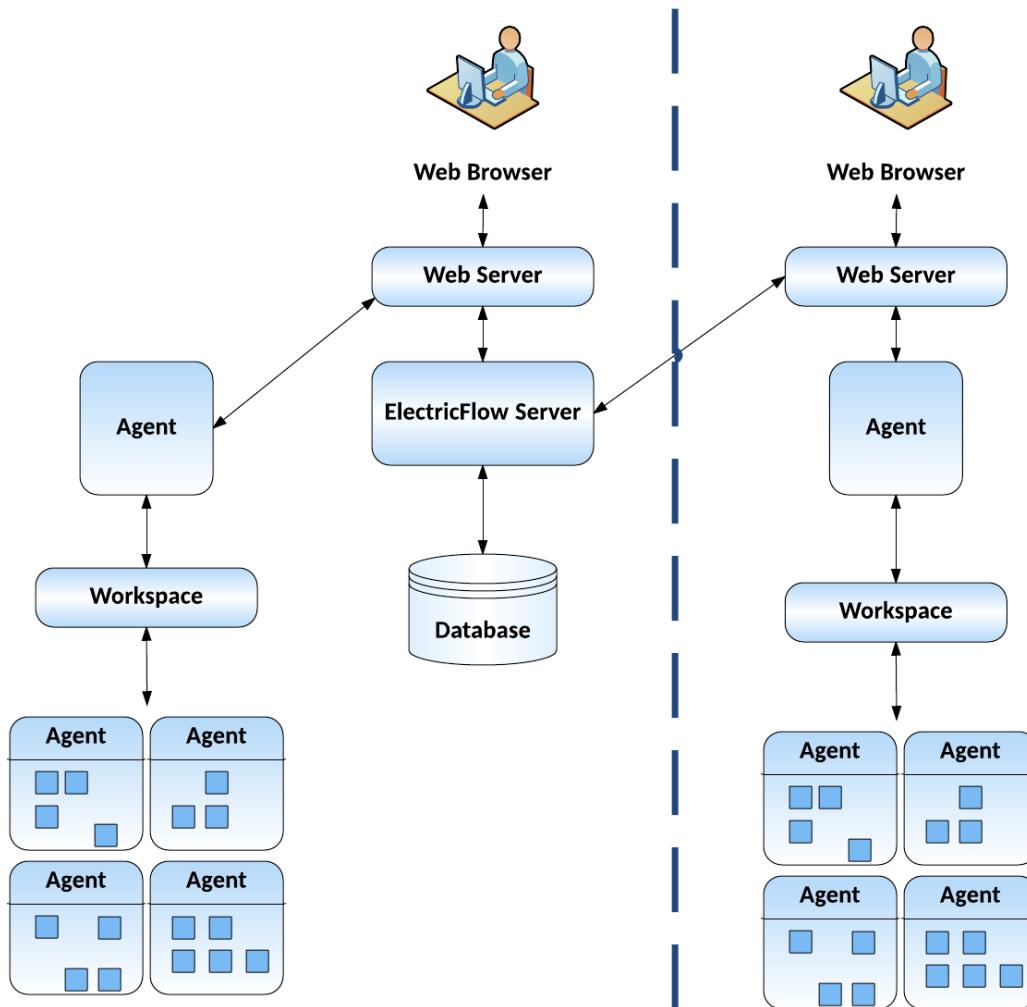
If you are only evaluating ElectricFlow, the ElectricFlow software, the database, the ElectricFlow server, the web server, and the repository server can reside on the same machine.

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

- Proxy (universal) resources
- Remote database
- Multiple remote web servers
- Multiple remote repository servers
- Configurations designed specifically for *failover*

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 a server machine available to ElectricFlow for running steps. 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*.

- 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.

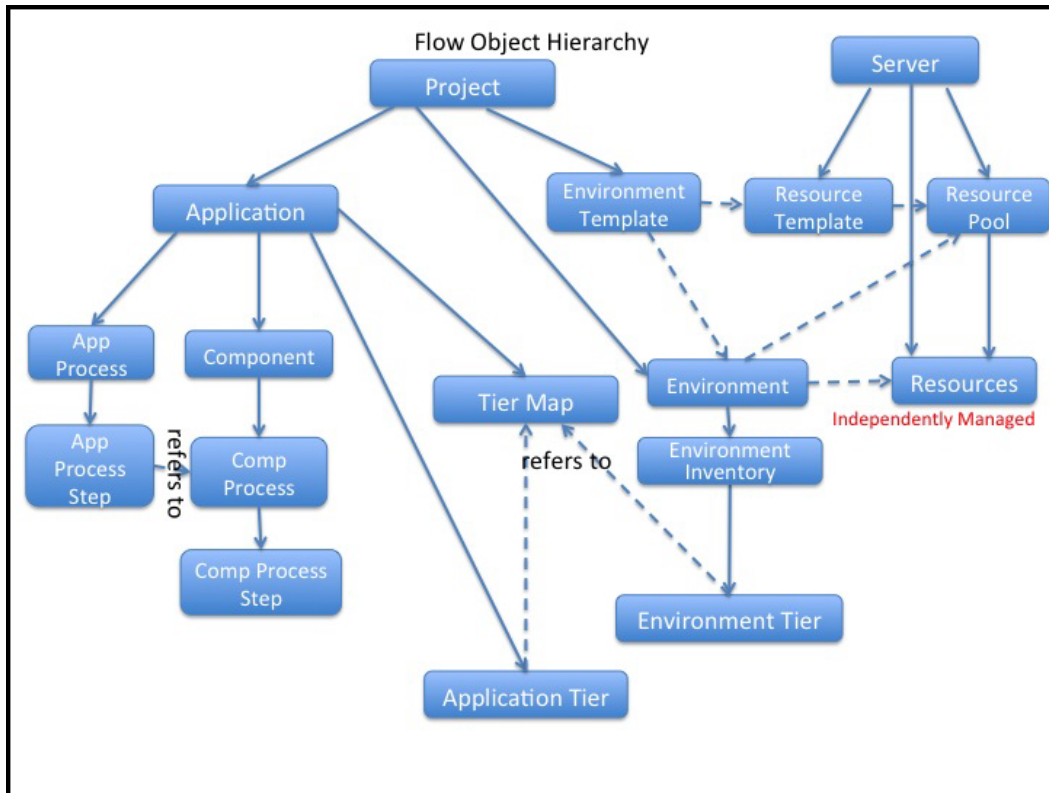
After setting resources, procedures, and schedules, ElectricFlow automatically runs the procedures that you created using these objects and facilities:

- **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 retrieve 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.

- *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 or change their priorities.
- *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.



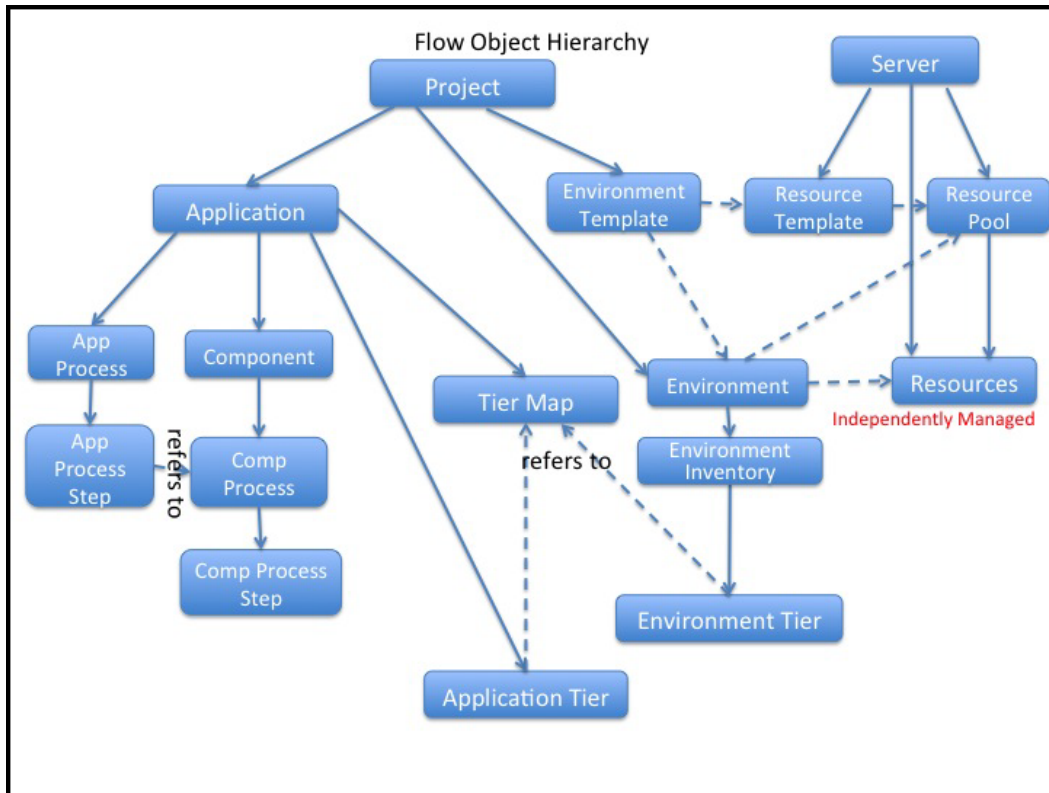
For more information about the ElectricFlow objects, concepts, and features in this topic, go to the [ElectricFlow Glossary](#) on page 18

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. <i>Run</i> is a synonym of <i>deploy</i> 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 Glossary](#) on page 18

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 Glossary](#) on page 18

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 797
RESTful APIs	Using the ElectricFlow RESTful API on page 788
Perl APIs	Using ectool on page 41 Using Perl (ec-perl) on page 42 Using API Commands in Javascript on page 50

ElectricFlow Glossary

These objects and concepts apply to ElectricFlow.

Object	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 <i>access control entries</i> (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.
access control entry	An access control list (ACL) determines if a particular user can perform a particular operation on a specified object. The list contains <i>access control entries</i> (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.
access control list	An access control list (ACL) determines if a particular user can perform a particular operation on a specified object. The list contains <i>access control entries</i> (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	See access control entry on page 18.
ACL	See access control list on page 18.
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. Note: When the ElectricFlow server and agents are installed on different hosts, make sure that the configuration for each agent specifies the Domain Name System (DNS) server. See the Web Interface Help > Resources topic for more information.

Object	Description
API	<p>Application Programming Interface.</p> <p>ElectricFlow supports these APIs based on the higher-order to the lower-order programming constructs:</p> <ul style="list-style-type: none"> • DSL methods—You create scripts and templates without using API commands to execute the script. • RESTful APIs—You go to the RESTful API URI and enter the appropriate information on the API UI to execute to request. • Perl APIs—You access the command-line interface through ectool or ec-perl and enter the correct command syntax to execute the command.
application	<p><i>Application</i> can refer to one of these entities:</p> <ul style="list-style-type: none"> • Within ElectricFlow, you model and deploy (run) an application to build, test, deploy, and release your software for continuous delivery across different pipelines. An application consists of application tiers, components in these tiers, component processes, and application processes. Before running an application in ElectricFlow, you must map its application tiers to corresponding environment tiers where the application will run. • Outside of ElectricFlow, application refers to the software or application that you want to deploy, where the deployment includes build, test, installation, implementation, deploy, and release processes. This is the end product of using ElectricFlow.
Application Inventory	Where ElectricFlow shows the status of an application as it runs and the results of previous runs. It includes when the application ran, how long it took, whether the application ran successfully or not, error messages, and links to more information for troubleshooting.
application process	A group of steps, actions, or component processes taken when the ElectricFlow application is deployed. You can reuse and rerun the process more than once.
application tier	A logical grouping of components in an ElectricFlow application, which must have one or more tiers with components. A tier can have one or more components.
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.
artifact repository	See the Artifact Management > Artifact objects > Repository topic in the <i>ElectricFlow User Guide</i> for more information.

Object	Description
artifact version	An artifact version is a collection of 0 to N files that were published to an artifact repository.
automation platform	See ElectricFlowplatform .
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.
build-test automation	Executing automated build and test processes, resulting in reduced costs, increase quality, reliability and traceability, and accelerated time to market. Build-test automation is a key enabler of continuous integration (CI).
change history	A record of the historical states of the system and the changes between them over time.
Change Tracking	How ElectricFlow monitors the changes to tracked objects, including applications, procedures, workflows, workspaces, resources, and project-owned components (such as libraries), and how it records a <i>change history</i>
component	<p>An object that is based on a specific version of an artifact and is defined in an application. Artifacts are defined and managed in the ElectricFlow automation platform.</p> <p>A component is the result of running an ElectricFlow application and has details, properties, and access control settings.</p> <p>It can be used by other ElectricFlow applications, or it can be the part of the continuous delivery solution. You add a component to an application tier.</p>
component process	A group of steps or actions taken on a component when the ElectricFlow application is deployed. You can reuse and rerun these processes more than once.
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 Delivery	<p>Continuous Delivery (CD) is a set of practices and methodologies in software development designed to improve the process of software delivery and ensuring reliable software releases.</p> <p>The goal of Continuous Delivery is to keep software release-ready, and enable a repeatable, reliable way to deploy software to any environment.</p>

Object	Description
Continuous Integration	<p>Using continuous integration means a build is launched every time code changes are checked into a Source Control Management (SCM) system.</p> <p>The ElectricSentry component is the engine for continuous integration, while the CI Continuous Integration Dashboard is the front-end user interface for ElectricSentry. The Continuous Integration Manager (CI Manager) provides a front-end user interface (dashboard) for creating, managing, and monitoring continuous integration builds. For example, using your preferred SCM, you can run a procedure to build your software every time you check in code.</p>
credential	<p>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: <i>stored</i> or <i>dynamic</i>.</p>
custom property	<p>Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner and behave in every way like an intrinsic object-level property with one exception: they are not created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution.</p> <p>Custom properties in a property sheet can be one of two types: <i>string</i> property or a <i>property sheet</i> 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.</p>
deploy	<p><i>Deploy</i> can refer to one of these activities:</p> <ul style="list-style-type: none"> • Within ElectricFlow—Running the application that you modeled in ElectricFlow. The result is your software, application, or system, such as a WAR file, database, or configuration. <p>You model an application and an environment, define component and application processes, map application tiers to environment tiers, and run the application in the environment.</p> <p>Note: Within ElectricFlow, the terms <i>deploy</i> and <i>run</i> are synonymous. When deploying an application in ElectricFlow, you are actually running it to produce your software or application..</p> <ul style="list-style-type: none"> • Outside of ElectricFlow—Running all the processes, procedures, or actions to develop and deploy your software in the appropriate environment, where the deployment includes build, test, installation, implementation, deploy, and release phases.

Object	Description
deploying (running) applications	<p>You can deploy applications one of these ways:</p> <ul style="list-style-type: none"> • Full deploy–The system deploys all the objects (including application processes, components, and artifacts) in the application. • Smart deploy–The system deploys only the artifacts that have not been deployed to a resource or specific versions of the artifacts or have not been deployed to new resources. • Partial deploy–The system deploys only objects that you select. • Partial deploy with specific artifact versions–The system deploys only the artifacts with selected versions. • Schedule–Create schedules to run applications on a one-time, daily, weekly, or monthly basis. • Snapshot–Select a snapshot to deploy.
deployment automation	<p>Executing automated application deployment processes, resulting in reduced costs, increase quality, reliability and traceability, and accelerated time to market. Deployment automation is a key enabler of Application Release Automation (ARA) and continuous delivery (CD).</p>
description	<p>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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p>
diagnostic extract	<p>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.</p> <p>The ElectricFlow <i>postp</i> postprocessor uses filenames like <code>diag-2770.xml</code>, where "2770" is the unique identifier for the step. Other postprocessors you may use can have a different filename configuration.</p>
domain specific language	<p>A domain specific language (DSL) is a programming language designed for a specific domain to accomplish specific tasks. You do not need to know how to use the API for the domain.</p> <p>The ElectricFlow DSL allows you to create scripts or templates for all the operations that you can do using the UI, the RESTful API, or the Perl API.</p>
DSL	See domain specific language on page 22.
dynamic credential	<p>Dynamic credentials are captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded.</p>

Object	Description
dynamic environment	<p>An environment that is automatically created on an on-demand basis,when you deploy an application in ElectricFlow.</p> <ul style="list-style-type: none"> • An environment with only provisioned cloud resources. • An environment with only static resources added to an environment template. • An environment with provisioned cloud resources and static resources <p>The provisioned cloud resources are spun up when you are ready to deploy the application. The static resources that are part of your system or network (such as servers, databases, and agent machines).</p>
dynamic resource pools	Resource pools that are created by provisioning a resource or environment template in ElectricFlow.
dynamic resources	Resources in the cloud that you can provision before the application is deployed. You can also group these resources into dynamic cloud resource pools. You can define these resources in resource templates.
ec-perl	<p><i>ec-perl</i> is a small wrapper program installed as part of ElectricFlow. When the <i>ec-perl</i> wrapper runs, it sets up the environment, finds, and calls the ElectricFlow copy of Perl, passing all of its parameters to Perl.</p> <p>You can run <i>ec-perl</i> when using the ElectricFlow Perl API.</p>
ectool	<p><i>ectool</i> 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 <i>ectool</i>.</p> <p>You can run <i>ectool</i> when using the ElectricFlow Perl API.</p>
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.
ElectricFlow platform	Software that provides domain-specific capabilities to automate some or all phases of your software delivery process, including the build, test, integrate, deploy, and release processes.automatically runs tasks and procedures and manages the objects in them (formerly known as <i>ElectricCommander</i>).
ElectricSentry	<p>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 ElectricFlow plugin named ECSCM and in the Electric Cloud project.</p> <p>Note: The CI Continuous Integration Dashboard is the front-end user interface for ElectricSentry.</p>

Object	Description
email configuration	Before you can send an email notifier, you must set up an 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.
environment	Within the ElectricFlow system, the location to which a resource is assigned and where jobs, procedures, application, or pipelines run. See also <i>dynamic environment</i> and <i>static environment</i> . Before running an application in ElectricFlow, you must map its application tiers to corresponding environment tiers where the application will run.
Environment Inventory	How ElectricFlow represents the status of an application as it runs in a specific environment at any point in time during the life cycle of your software. It tracks the application processes as they run and the results of previous runs, when the application ran, how long it took, the versions of the deployed software artifacts, the resources used, error messages, and links to more information for troubleshooting.
environment template	A template defining an environment that can be spun up when the application is deployed. The template details include the environment name, its description, the environment tiers, and the resources assigned to the environment tiers. You can add one or more static resources to an environment tier. When adding resource templates to a tier, you can add only one resource template and then enter the number of dynamic resources to provision.
environment tier	A logical grouping of resources in an ElectricFlow environment, which must have one or more tiers with resources. A tier can have more than one resource.
event log	Log information about system events. See log on page 27 for more information.
Everyone	A special intrinsic access control group that includes all users.
filter	Two filter categories: <ul style="list-style-type: none"> • 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).

Object	Description
formal parameter	<p>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.</p> <p>Formal parameters are different from <i>actual parameters</i>. Formal parameters define the kinds of parameters a procedure is expecting, and actual parameters provide values to use at run-time.</p>
gateway	<p>To communicate with a resource, workspace, or artifact repository server in another zone, a <i>gateway</i> must be created.</p> <p>A gateway object contains two resource (agent) machines. For example, <i>GatewayResource1</i> and <i>GatewayResource2</i> are configured to communicate with the other. One gateway resource resides in the <i>source</i> zone and the other in the <i>target</i> zone.</p> <p>A gateway is bidirectional and informs the ElectricFlow server that each gateway machine is configured to communicate with a gateway machine in another zone.</p>
group	<p>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 <i>locally</i> in the ElectricFlow server.</p> <p>See the ElectricFlow Help > Web Interface Help > Users and Groups > Groups topic for more information.</p>
impersonation	<p>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).</p> <p>Impersonation is implemented using credentials.</p>
inheritance	<p>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.</p>
intrinsic property	<p>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.</p> <p>For example: Every project has a <i>description</i> property that can be referenced with a non-local property path such as <code>/projects/Examples/description</code>.</p>
Inventory Tracking	<p>How ElectricFlow tracks what is deployed for continuous delivery.</p> <p>ElectricFlow tracks this information at the application and environment levels. The environment inventory is more comprehensive than the application inventory. For more information, see the application inventory and environment inventory.</p>

Object	Description
job	<p>A job is the output associated with invoking a ElectricFlow procedure. A new job is created each time you run (execute) a procedure.</p> <p>This records the results of running a procedure. You can view the results of running the procedure and each step within the procedure. Tables are on the Job Details page to view or manage your results or to use for future reference.</p>
job configuration	<p>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 automation platform 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:</p> <ul style="list-style-type: none"> • From the Job Details page for a previously invoked job, click Save Configuration 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 Create 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 Save Configuration at the top of the page. Your saved configuration will be displayed on your Home page.
job name template	<p>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 may enter:</p> <pre> \${projectName}_\${increment /myproject/jobCounter}_\${timestamp} </pre> <p>which produces a name like:</p> <pre> projectFoo_1234_20140102130321 </pre> <p>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.</p>
jobs quick view	<p>A Jobs Quick View section is one of the facilities provided on the automation platform 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.</p>
job step	<p>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.</p>
job workspace	<p>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.</p>

Object	Description
launch pad	A starting point on the Home page from which you can go to the Applications List, Environment List, or automation platform to model the ElectricFlow application. You deploy this application to build, test, deploy, and release your software for continuous delivery.
local group	A group defined <i>inside</i> 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 on page 25 for more information.
local user	A user defined <i>inside</i> 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 on page 38 for more information.
log	<p>ElectricFlow provides a log for events generated anywhere in the system, including jobs and workflows.</p> <p>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.</p> <ul style="list-style-type: none"> • To see <i>only</i> 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 <i>only</i> 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 <i>only</i> 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", and 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.
matcher	A <i>matcher</i> controls the postp on page 29 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.

Object	Description
misfire policy	<p>A misfire policy allows you to manage how a schedule resumes in cases where the normal scheduled time is interrupted.</p> <p>Available options are:</p> <ul style="list-style-type: none"> • <code>skip</code> (all misfires are ignored and the job runs at the next scheduled time). • <code>run once</code> (after one or more misfires, the job runs at the soonest time that occurs within an active region). <p>See schedule on page 35 for more information.</p>
parameter	<p>A property value passed into a procedure when it is invoked (at <i>run</i> time), and used by the procedure to change its behavior. Two types of parameters are actual parameter on page 18 and formal parameter on page 25</p>
pipeline	<p>Pipelines easily model a reusable single pathway to production across multiple stages and environments and provide an aggregated view of all the activity.</p> <p>Use a pipeline when orchestrating a deployment consisting of more than one application or when orchestrating automations.</p> <p>Pipeline objects are split into two types: <i>Definition</i> objects and <i>Instance</i> objects.</p> <p>In this ElectricFlow release, you can model generic pipelines.</p>
pipeline management	<p>Executing automated end-to-end software development and delivery processes, resulting in reduced costs, increase quality, reliability and traceability, and accelerated time to market. Pipeline management is a key enabler of Application Release Automation (ARA) and Continuous Delivery (CD).</p>
pipeline definition	<p>Pipeline definition objects provide a template for a running pipeline instance.</p> <p>When you run the pipeline definition, ElectricFlow creates a new pipeline object that represents the run-time instance of the pipeline definition.</p>
pipeline instance	<p>Each time a pipeline runs, ElectricFlow creates an instance of the pipeline.</p> <p>For each instance, ElectricFlow creates logs of what happened. You can view these pipeline details:</p> <ul style="list-style-type: none"> • Status • Time • Who • What • When • Evidence

Object	Description
plugin	<p>An add-on program used by ElectricFlow to integrate with third-party tools, custom dashboards, and unique user experiences based on roles.</p> <p>ElectricFlow is built with an extensible architecture, allowing easy development of plugins that include integrations with other tools, custom dashboards, and unique user experiences based on roles. Numerous plugins are installed during the ElectricFlow installation, which makes them transparent to the user.</p> <p>For example, in the Scenario 1 and Scenario 2 Help topics, you will configure a source control system and build an Ant step, both of which use plugin technology.</p>
polling frequency	<p>The <i>polling frequency</i> is how often the ElectricSentry continuous integration engine is set to look for new code check-ins.</p> <p>The default is set to every 5 minutes.</p>
pool	<p>Also known as <i>resource pool</i>.</p> <p>A pool is a collection of resources.</p> <p>If a step specifies a pool name as its resource, ElectricFlow can choose any available resource within that pool.</p>
postp	<p><i>postp</i> is a postprocessor included with ElectricFlow. <i>postp</i> uses regular expression patterns to detect interesting lines in a step log.</p> <p><i>postp</i> is already configured with patterns to handle many common cases such as error messages and warnings from <i>gcc</i>, <i>gmake</i>, <i>cl</i>, <i>junit</i>, and <i>cppunit</i>, or any error message containing the string "error."</p> <p><i>postp</i> also supports several useful command-line options, and it can be extended using "matchers" to handle environment-specific errors.</p> <p>See matcher on page 27 for more information.</p>
postprocessor	<p>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.</p> <p>ElectricFlow includes a standard postprocessor called <i>postp</i> for your use and you can "extend" <i>postp</i>.</p> <p>See matcher on page 27 for more information.</p>
preflight build	<p>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.</p>

Object	Description
privileges	<p>ElectricFlow supports four privilege types for access control on page 18 and security for each object:</p> <ul style="list-style-type: none"> • 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 <i>project</i>), 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	<p>A procedure defines a process to automate one or more steps. A procedure is the ElectricFlow unit you execute (<i>run</i>) 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.</p> <p>During deployment automation, you can also call a procedure in an application or component process.</p>
process branching	How to run job steps in an application or component process on a conditional basis in ElectricFlow.
process type	<p>Select one of the following parameters to configure how Inventory Tracking works on an application or component process in ElectricFlow:</p> <ul style="list-style-type: none"> • Deploy—Select this to enable Inventory Tracking. The ElectricFlow server tracks artifacts deployed to environments. This is the default. • Undeploy—Select this to configure the ElectricFlow automation platform to remove the environment inventory record after the first job step in a component process runs successfully. • Other—Select this to disable Inventory Tracking.
procedure	<p>A procedure contains a group of steps, each performing a task to do the work you define. You can define as many steps as you need for each procedure. Procedures are reusable indefinitely or can be modified whenever you choose.</p>

Object	Description
project	<p>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.</p> <p>ElectricFlow uses a project as a container for related procedures or any procedures you choose to group together. Your project can contain as many procedures as you decide are necessary.</p> <p>Projects have two purposes:</p> <ul style="list-style-type: none">• 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	<p><i>Project principal</i> 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.</p>

Object	Description
property	<p>A property is a <code>name-value</code> 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. Two types of properties: <i>intrinsic</i> and <i>custom</i>.</p> <p>ElectricFlow provides intrinsic properties and allows you to create custom properties.</p> <p>Note: Intrinsic properties are case-sensitive. Custom properties, like all other object names in the ElectricFlow system, are case-preserving, but not case-sensitive.</p> <ul style="list-style-type: none"> 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 <code>Description</code> property that can be referenced with a non-local property path such as <code>/projects/Examples/description</code>. Custom properties Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner, and behave in every way like an intrinsic object-level property with one exception: they are not created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution.
property sheet	A property sheet is a collection of properties that can be nested to any depth. The property value can be a string or a nested property sheet. Most objects have an associated "property sheet" that contains custom properties created by user scripts.
proxy agent	A proxy agent is an agent on a supported Linux or Windows platform, used to proxy commands to an otherwise unsupported agent platform. Proxy agents have limitations, such as the inability to work with plugins or communicate with ectool commands.
proxy resource	This resource type requires SSH keys for authentication. You can create proxy resources (agent on page 18 and target) 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 <i>publish</i> operation for an artifact version.

Object	Description
quiet time	<p>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.</p>
RESTful API	<p>An API for web services based on the REST (REpresentational State Transfer), a simple stateless architecture.</p> <p>ElectricFlow has a RESTful API with POST, DELETE, GET, and PUT operations that are sent as HTTP requests. You do not need to have in-depth knowledge of the API to use RESTful API commands.</p>
report	<p>Reports display your information graphically for review or to show trends and other information.</p> <p>ElectricFlow provides multiple reports and custom report capabilities to help you manage your build environment.</p> <ul style="list-style-type: none"> • Real-time reports have filtered view of your workload in real-time. • Build reports are summary reports produced at the end of a build and attached to the job. • Batch reports are summaries of your build environment with trends over time: <ul style="list-style-type: none"> • Default Batch reports are automatically installed during ElectricFlow installation and scheduled to run daily (such as Cross Project Summary, Variant Trend, Daily Summary, Resource Summary, Resource Detail). • In Optional Batch reports, you can configure, rename, and schedule batch reports to fit your requirements (such as Category Report, Procedure Usage Report, Count Over Time Report, Multiple Series Reports). • In custom reports, you choose how to create and add at any time.
repository or repository server	<p>A <i>repository</i> 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.</p> <p>Connection information is stored in the repository object on the ElectricFlow server.</p> <p>The artifact repository is a machine where artifact versions are stored in either uncompressed <code>tar</code> archives or compressed <code>tar-gzip</code> archives. The repository server is configured to store artifact versions in a directory referred to as the repository <i>backing store</i>.</p> <p>By default, the backing store is the <code><datadir>/repository-data</code> directory in the repository installation.</p>

Object	Description
resource	<p>An agent machine that is configured to communicate with ElectricFlow and where job steps can be executed.</p> <p>Steps can run on any resource or resource pool that you define.</p> <p>Resources can be grouped into a pool to increase the processing speed by assigning steps to less loaded members of the pool. Pools can also be used to ensure the resources you need are available only to a specific group of developers.</p> <p>ElectricFlow supports two types of resources:</p> <ul style="list-style-type: none"> • 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. <p>See also <i>dynamic resource</i> and <i>static resource</i>.</p> <p>During deployment automation, you define resources in environments during deployment automation.</p>
resource template	<p>A template with the required information to provision and later spin up cloud resources on an on-demand basis. You set the cloud provider and configuration management details in a resource template. In an environment template, you define environment tiers and can assign resource templates to the environment tiers. You can add, edit, or remove resource templates.</p>
RESTful API	<p>A method of communication between a Web-based client and server that employs representational state transfer (REST) constraints.</p>
run	<p><i>Run</i> can refer to these activities:</p> <ul style="list-style-type: none"> • Within ElectricFlow—Deploying the application that you modeled in ElectricFlow. The result is your software, application, or system, such as a WAR file, database, or configuration. <p>You model an application and an environment, define component and application processes, map application tiers to environment tiers, and run the application in the environment.</p> <p>Note: Within ElectricFlow, the terms <i>deploy</i> and <i>run</i> are synonymous. When deploying an application in ElectricFlow, you are actually running it to produce your software or application..</p> <ul style="list-style-type: none"> • Outside of ElectricFlow—Deploying all the processes, procedures, or actions to develop and deploy your software in the appropriate environment, where the deployment includes build, test, installation, implementation, deploy, and release phases.

Object	Description
schedule	<p>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:</p> <ul style="list-style-type: none"> • Standard—A standard schedule defines when your procedure will run. You can control the hour and day that your procedure runs or use a trigger to run a procedure each time a particular event occurs. • Continuous Integration • 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	<p>One type of shortcut is part of the automation platform 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.</p> <p>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.</p>
snapshot	A version of an application with specific artifact versions and the state of the application at any point in time.
state	Workflows always have a single active <i>state</i> . 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.
state definition	<p>Workflow objects are split into two types: <i>Definition</i> objects and <i>Instance</i> 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.</p> <p>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.</p> <p>Note: We omit the "Instance" qualifier for brevity in the API and the UI.</p> <p>Each workflow can contain one or more state objects. Defining states for a workflow is analogous to defining steps for a procedure.</p>

Object	Description
static environment	An environment with resources that are in your system or network, such as servers, databases, and agent machines. You model this environment and assign static resources to it before deploying the application.
static resource	A resource located in your system or network, not in the cloud. Servers, databases, and agent machines are examples of static resources.
step	<p>A step is part of a procedure or process. Using almost any kind of script, you can define a specific task for a step to perform. Steps can run in parallel for faster processing. Steps can be modified or copied to other procedures whenever you choose.</p> <p>Each step specifies a command to execute on a particular resource or a subprocedure (nested procedure) to invoke. Commonly created steps include:</p> <ul style="list-style-type: none"> • Command—This step invokes a <code>bat</code>, <code>cmd</code>, <code>shell</code>, <code>perl</code> 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: <ul style="list-style-type: none"> • Publish or retrieve artifact version • Send Email • Various SCM step types • Build tools
stored credential	<p><i>Stored credentials</i> are given a name and stored in encrypted form in the database. Each project contains a list of stored credentials it owns.</p> <p>These credentials are managed from the Project Details page.</p>
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	<p>A mechanism used to include property values in step commands and elsewhere.</p> <p>For example, if a step command is specified as <code>echo \${status}</code>, and when the step executes there is a property named <code>status</code> with value <code>success</code>, the actual command executed will be <code>echo success</code>.</p>
system object	<p>This is a special object whose access control lists are used to control access to some ElectricFlow internals.</p> <p>System objects are: <code>admin</code>, <code>artifactVersions</code>, <code>directory</code>, <code>emailConfigs</code>, <code>forceAbort</code>, <code>licensing</code>, <code>log</code>, <code>plugins</code>, <code>priority</code>, <code>projects</code>, <code>repositories</code>, <code>resources</code>, <code>server</code>, <code>session</code>, and <code>workspaces</code>.</p>

Object	Description
tag	<p>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.</p> <p>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.</p>
tier map	<p>The mapping of the application tiers to the corresponding environment tiers where the application will run.</p> <p>To run an application, you map one application tier to one or more environment tiers and must have at least tier map.</p>
transition	<p>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:</p> <ul style="list-style-type: none"> • 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. <p>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.</p> <ul style="list-style-type: none"> • 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.
transition definition	<p>How a workflow transitions from one state to another.</p> <p>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.</p> <p>Note: We omit the "Instance" qualifier for brevity in the API and the UI.</p> <p>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.</p> <p>You can define one or more transitions for each state, depending on which transition options you want to apply to a particular state.</p>

Object	Description
user	<p>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.</p> <p>See local user on page 27 for more information.</p>
workflow	<p>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.</p> <p>A workflow contains <i>states</i> and <i>transitions</i> 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.</p> <p>Workflow objects are split into two types: <i>Definition</i> objects and <i>Instance</i> 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.</p>
workflow definition	<p>Workflow objects are split into two types: <i>Definition</i> objects and <i>Instance</i> 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.</p> <p>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.</p> <p>Note: We omit the "Instance" qualifier for brevity in the API and the UI.</p>
workflow name template	<p>This is the template used to determine the default name of jobs launched from the workflow definition.</p> <p>For example:</p> <pre> \$[projectName]_\$/[increment /myproject/workflowCounter]_ \$[timestamp] </pre> <p>(substitute your values for the names above)</p> <p>Produces a workflow name like:</p> <pre> projectName_123_20140102130321 </pre>

Object	Description
workspace	<p>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.</p> <p>ElectricFlow provides a default area on the disk that it can use for working files and results. This disk area is called a job workspace.</p> <p>A job step can create whatever files it needs in its workspace, such as step logs, and ElectricFlow automatically places these files in the workspace.</p> <p>A single workspace can be shared by all steps in a job, but it is possible for different steps in a job to use different workspaces.</p> <p>The location of the job step workspace is displayed on the Job Details page for the job under the "Details" for the step.</p>
workspace root	<p>A workspace root is a directory in which ElectricFlow allocates job workspace directories.</p> <p>Each workspace root has a logical name used to refer to it in steps and procedures.</p>
zones	<p>A zone is a way to partition a collection of agents to secure them from use by other groups.</p> <p>For example, you might choose to create a developers zone, a production zone, and a test zone. The agents in one zone cannot directly communicate with agents in another zone.</p> <p>A <i>default</i> zone is created during ElectricFlow installation.</p> <p>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).</p>

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 manage artifacts.
- Create and manage object properties.
- Create and manage resources.
- Create workflows and add resources to them.
- Create and call procedures.
- Model and deploy applications.
- Model and run pipelines.

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](#)
- [Using ec-perl](#)

- [Common global options](#)
- [The Batch API](#)
- [Installing ElectricFlow Perl modules into your Perl distribution](#)
- [Installing Perl modules into the ElectricFlow Perl distribution](#)
- [Using Perl APIs in Javascript](#)

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 bldglserver login "Ellen Ernst" "eel23"
```

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
<code>--help</code>	Display an online version of <code>ectool</code> commands with a short description. Displays command information if followed by a command name.
<code>--version</code>	Display the <code>ectool</code> version number.

Global Arguments	Description
<code>--server <hostname></code>	<p>ElectricFlow server address. Defaults to the <code>COMMANDER_SERVER</code> 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 <code>localhost</code>.</p> <p>Note: If you are using multiple servers, we recommend using the <code>server</code> option to ensure the correct server is specified for your task. For example, if you are using the <code>import</code> API, the <code>server</code> option may be particularly important.</p> <p>Do not use in a step context: we recommend that steps running <code>ectool</code> or Perl scripts should <i>never</i> provide the <code>server</code> 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.</p> <p>In a step context, <code>ectool</code> 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. <code>ectool</code> / 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.</p> <p>Generally, the issue is that the server publicizes its name as a fully-qualified domain name and <code>ectool</code> / 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 <code>ectool</code> so they no longer include the server-name, and <code>ectool</code> will default to the server-name that the server provided.</p>
<code>--port <port></code>	HTTP listener port on the ElectricFlow server. Defaults to port 8000.
<code>--securePort <secureport></code>	HTTPS listener port on the ElectricFlow server. Defaults to port 8443.
<code>--secure <0 1></code>	<p>Use HTTPS to communicate with the ElectricFlow server.</p> <p>Note: Certain requests (for example, <code>login</code>, <code>createUser</code>, and <code>modifyUser</code>) automatically use HTTPS because passwords are being sent, which means it is not necessary to specify <code>secure</code> for those APIs. Defaults to 1.</p>
<code>--timeout <s></code>	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.
<code>--retryTimeout <s></code>	This is a separate timer, independent of the <code>retry</code> 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.
<code>--retry <0 1></code>	Retry the request if it times out based on the "timeout" value. Default is "0" and should rarely be changed.

Global Arguments	Description
<code>--user <username></code>	Use the session associated with the user. Defaults to the user who last logged in.
<code>--service <spn></code>	Specify the service principal name to use for Kerberos. Defaults to HTTP@host.domain.
<code>--setDefault <0 1></code>	Use the current session as the default for subsequent invocations. Defaults to 1.
<code>encoding <charEncoding></code>	Use the specified encoding for input/output. For example, for charEncoding, enter UTF-8, cp 437, and so on. Default is autodetected.
<code>--dryrun</code>	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 <code>--user</code> or <code>--server</code> options.
<code>--silent</code>	Suppresses printing the result. For example: <code>ectool --silent createResource foo</code> 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.
<code>--valueOf</code>	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 <code>--valueOf <path></code> option evaluates the XML result and emits the value of that node to satisfy such use cases. For example: <code>\$ ectool --valueOf '//version' getServerStatus</code> returns only "4.1.0.48418".
<code>--format <format></code>	Specifies the response format. Must be one of 'xml' or 'json'. Defaults to 'xml'. For example, you might specify: <code>ectool --format json setProperty summary hello</code>
<code>--ignoreEnvironment</code>	Force ectool to ignore <code>COMMANDER_ENV</code> 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:
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\lib',
  sitearchexp => 'X:\perl\lib',
#sitelibexp => 'C:\Program Files\Electric Cloud\ElectricCommander\perl\site\lib',
  sitelibexp => 'X:\perl\lib',
```

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

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' } )).project.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":{"resourceId":"ceecf5e5-2d0d-11e4-8888-005056330afe","resourceName":"local","agentState":{"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_64 GNU/Linux","hostPlatform":"linux","message":"The agent is alive","pingToken":"1409049660","protocolVersion":"6","state":"alive","time":"2014-08-26T10:43:25.802Z","version":"5.0.3.76444"},"createTime":"2014-08-26T10:43:25.617Z","description":"Local resource created during installation.","hostName":"qa-ub1210-64-2.electric-cloud.com","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_64 GNU/Linux","hostPlatform":"linux","lastModifiedBy":"project: zebra","lastRunTime":"2014-08-26T10:50:23.786Z","modifyTime":"2014-08-26T10:50:23.786Z","owner":"admin","port":"7800","proxyPort":"","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 40](#)" 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
<code>createApplication</code>	Creates a new application for a project.
<code>deleteApplication</code>	Deletes an application.
<code>getApplication</code>	Retrieves an application by name.
<code>getApplications</code>	Retrieves all applications in a project.
<code>modifyApplication</code>	Modifies an existing application.

Application Tiers

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

Artifact Management

Commands	Description
<code>addDependentsToArtifactVersion</code>	Adds an artifact version query to an existing artifact. Dependent artifact versions are retrieved when the parent artifact version is retrieved.
<code>cleanupArtifactCache</code>	Deletes stale artifact versions from an artifact cache. A "stale artifact version" is one whose metadata was previously deleted from the ElectricFlow server.
<code>cleanupRepository</code>	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
<code>createArtifact</code>	Creates a new artifact.
<code>createArtifactVersion</code>	Creates a new artifact version.
<code>createRepository</code>	Creates a repository for one or more artifacts.
<code>deleteArtifact</code>	Deletes an existing artifact element and all artifact versions.
<code>deleteArtifactVersion</code>	Deletes artifact version metadata from the ElectricFlow database. (This API call does not delete or remove artifacts stored on the repository machine.)
<code>deleteRepository</code>	Deletes artifact repository metadata from the ElectricFlow database. (This API call does not delete or remove artifacts stored on the repository machine.)
<code>findArtifactVersions</code>	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 <code>findArtifactVersions</code> command.
<code>getArtifact</code>	Retrieves an artifact by its name.
<code>getArtifacts</code>	Retrieves all artifacts in the system.
<code>getArtifactVersion</code>	Retrieves an artifact version by its name.
<code>getArtifactVersions</code>	Retrieves all artifact versions in the system, filtered by artifact name, retriever job ID, and/or retriever job step ID.
<code>getManifest</code>	Retrieves the manifest for a specified artifact version, which includes a list of files and directories in the artifact version, plus its checksum file.
<code>getRepositories</code>	Retrieves all artifact repository objects known to the ElectricFlow server.
<code>getRepository</code>	Retrieves an artifact repository by its name.
<code>getRetrievedArtifacts</code>	Retrieves artifacts during a job.
<code>modifyArtifact</code>	Modifies an existing artifact.
<code>modifyArtifactVersion</code>	Modifies an existing artifact version.
<code>modifyRepository</code>	Modifies an existing artifact repository.

Commands	Description
<code>moveRepository</code>	Moves an artifact repository in front of another, specified repository or to the end of the list.
<code>publishArtifactVersion</code>	Publishes an artifact version to an artifact repository.
<code>removeDependentsFromArtifactVersion</code>	Removes a list of dependent artifact versions from an existing artifact version.
<code>resolveRoute</code>	Resolves the network route to an artifact repository.
<code>retrieveArtifactVersions</code>	Retrieves the most recent artifact version, including its dependents, from an artifact repository.
<code>updateArtifactVersion</code>	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
<code>getDeploymentHistoryItems</code>	Retrieves all the deployment history items for a specific environment.
<code>getEntityChange</code>	Retrieves the entity changes.
<code>getEntityChangeDetails</code>	Retrieves the differences between entities.
<code>pruneChangeHistory</code>	Prunes obsolete-for-days data from the Change History tables.
<code>revert</code>	Reverts the state of the object to a previous state.
<code>searchEntityChange</code>	Searches for entity changes.

Components

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

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

Credential Management

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

Database Configuration

Commands	Description
<code>getDatabaseConfiguration</code>	Retrieves the current database configuration.

Commands	Description
<code>setDatabaseConfiguration</code>	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.
<code>validateDatabase</code>	Performs consistency checks on the database.

Directory Provider Management

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

Dynamic Environments

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

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

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

Email Configuration and Management

Commands	Description
<code>createEmailConfig</code>	Creates a new email configuration.
<code>deleteEmailConfig</code>	Deletes an email configuration.

Commands	Description
<code>getEmailConfig</code>	Retrieves an email configuration by name.
<code>getEmailConfigs</code>	Retrieves all email configurations.
<code>modifyEmailConfig</code>	Modifies an existing email configuration.

Email Notifiers Management

Commands	Description
<code>createEmailNotifier</code>	Creates an email notifier on an object specified by an <code>emailNotifierSelector</code> .
<code>createEventSubscription</code>	Creates a list of event subscriptions.
<code>deleteEmailNotifier</code>	Deletes an email notifier from a property sheet container.
<code>deleteEventSubscription</code>	Deletes a list of event subscriptions.
<code>getEmailNotifier</code>	Retrieves an email notifier from a property sheet container.
<code>getEmailNotifiers</code>	Retrieves all email notifiers defined for the specified property sheet container.
<code>modifyEmailNotifier</code>	Modifies an email notifier in a property sheet container specified by an <code>emailNotifierSelector</code> .
<code>modifyEventSubscription</code>	Modifies a list of event subscriptions.
<code>sendEmail</code>	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
<code>createEnvironment</code>	Creates a new environment.
<code>createEnvironmentInventoryItem</code>	Creates a new environment inventory item.

Commands	Description
<code>deleteEnvironment</code>	Deletes an environment.
<code>deleteEnvironmentInventoryItem</code>	Deletes an inventory item.
<code>getEnvironment</code>	Retrieves an environment by name.
<code>getEnvironmentApplications</code>	Retrieves a list of applications installed on the given environment.
<code>getEnvironmentInventory</code>	Retrieves a per-component grouped list of inventory items.
<code>getEnvironmentInventoryItem</code>	Retrieves an inventory item.
<code>getEnvironmentInventoryItems</code>	Retrieves all the inventory items for a given environment.
<code>getEnvironments</code>	Retrieves all environments in a project.
<code>modifyEnvironment</code>	Modifies an environment.
<code>modifyEnvironmentInventoryItem</code>	Modifies an existing environment inventory item.
<code>seedEnvironmentInventory</code>	Creates a new environment.

Environment Tiers

Commands	Description
<code>addResourcesToEnvironmentTier</code>	Adds resources to the specified environment tier.
<code>createEnvironmentTier</code>	Creates a new environment tier.
<code>deleteEnvironmentTier</code>	Deletes an environment tier.
<code>getEnvironmentTier</code>	Retrieves an environment tier by name.
<code>getEnvironmentTiers</code>	Retrieves all environment tiers in an environment.
<code>modifyEnvironmentTier</code>	Modifies an environment tier.
<code>removeResourcesFromEnvironmentTier</code>	Removes the given resources from the given environment tier.

Gateway and Zone Management

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

Job Management

Commands	Description
<code>abortAllJobs</code>	Aborts all running jobs.
<code>abortJob</code>	Aborts a running job.
<code>abortJobStep</code>	Aborts any type of step—command step or subprocedure step.
<code>completeJob</code>	Completes an externally managed job.
<code>completeJobStep</code>	Completes an externally managed job step.
<code>createJob</code>	Creates an externally managed job.
<code>createJobStep</code>	Creates a job step in an existing job.
<code>deleteJob</code>	Deletes a job from the ElectricFlow database.
<code>findJobSteps</code>	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.

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

Parameter Management

Commands	Description
<code>attachParameter</code>	Attaches a formal parameter to a step.

Commands	Description
<code>createActualParameter</code>	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.
<code>createFormalParameter</code>	Creates a new formal parameter for a procedure.
<code>deleteActualParameter</code>	Deletes an actual parameter.
<code>deleteFormalParameter</code>	Deletes a formal parameter.
<code>detachParameter</code>	Detaches a formal parameter from a step.
<code>getActualParameter</code>	Retrieves an actual parameter by its name.
<code>getActualParameters</code>	Retrieves all actual parameters from a job, job step, or step.
<code>getFormalParameter</code>	Retrieves a formal parameter by its name.
<code>getFormalParameterOptions</code>	Retrieves possible option values for a procedure's formal parameter in the procedure using the options script registered for it.
<code>getFormalParameters</code>	Retrieves all formal parameters from a procedure, schedule, or step.
<code>modifyActualParameter</code>	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.
<code>modifyFormalParameter</code>	Modifies an existing formal parameter.
<code>validateFormalParameters</code>	Validates input parameters for a procedure using the validation script registered for it.

Pipelines and Releases

Commands	Description
<code>abortPipelineRun</code>	Aborts a pipeline run.
<code>completeManualTask</code>	Completes the manual task.
<code>completeRelease</code>	Completes the Release.
<code>createDeployer</code>	Creates a new Deployer task for a project or a Release.

Commands	Description
<code>createDeployerApplication</code>	Adds a Deployer application for a Release.
<code>createDeployerConfiguration</code>	Adds the Deployer configuration to the Deployer application.
<code>createNote</code>	Creates a new note.
<code>createPipeline</code>	Creates a new pipeline for a project.
<code>createRelease</code>	Creates a new Release for a project.
<code>createStage</code>	Creates a new stage in a pipeline for a project.
<code>createTask</code>	Creates a new task for a task container.
<code>deleteNote</code>	Deletes a note associated with an entity.
<code>deletePipeline</code>	Deletes a pipeline in a project.
<code>deletePipelineRun</code>	Deletes a pipeline runtime.
<code>deleteRelease</code>	Deletes a Release.
<code>deleteStage</code>	Deletes a stage in a project.
<code>deleteTask</code>	Deletes a task in a task container.
<code>getDeployerApplication</code>	Retrieves the application used in the Release by name.
<code>getDeployerApplications</code>	Retrieves all the applications in a Release.
<code>getDeployerConfiguration</code>	Retrieves a Deployer configuration.
<code>getDeployerConfigurations</code>	Retrieves all the configurations in the Deployer.
<code>getNote</code>	Retrieves a note associated with an entity.
<code>getNotes</code>	Retrieves all the notes associated with an entity.
<code>getPipeline</code>	Retrieves a pipeline by its name.
<code>getPipelineRuntimeDetails</code>	Retrieves pipeline runtime details.
<code>getPipelineRuntimes</code>	Retrieves pipeline runs
<code>getPipelineStageRuntimeDeployerTasks</code>	Returns the list of Deployer tasks and their details to be displayed in Pipeline Run Details page.

Commands	Description
<code>getPipelineStageRuntimeTasks</code>	Retrieves the list of pipeline stage tasks and the details about them that are displayed in the pipeline run view.
<code>getPipelines</code>	Retrieves all the pipelines.
<code>getRelease</code>	Retrieves a Release by name.
<code>getReleaseInventory</code>	Retrieves inventory artifacts created in a Release.
<code>getReleases</code>	Retrieves all releases.
<code>getStage</code>	Retrieves a stage by its name.
<code>getStages</code>	Retrieves all the stages for a pipeline.
<code>getTask</code>	Retrieves a task by its name.
<code>getTasks</code>	Retrieves a task by name.
<code>getWaitingTasks</code>	Retrieves a list of all the stage tasks that are currently waiting on manual tasks in pipeline run view.
<code>modifyDeployer</code>	Modifies an existing Deployer.
<code>modifyDeployerApplication</code>	Modifies the Deployer application associated with a Release.
<code>modifyDeployerConfiguration</code>	Modifies a Deployer configuration associated with a Deployer application.
<code>modifyNote</code>	Modifies a note associated with an entity.
<code>modifyPipeline</code>	Modifies an existing pipeline.
<code>modifyRelease</code>	Modifies an existing Release.
<code>modifyStage</code>	Modifies an existing stage.
<code>modifyTask</code>	Modifies an existing task.
<code>removeDeployerApplication</code>	Removes a Deployer application for a Release.
<code>removeDeployerConfiguration</code>	Removes a Deployer configuration associated with a Deployer application.
<code>runPipeline</code>	Runs the specified pipeline.
<code>startRelease</code>	Starts a Release.

Commands	Description
<code>validateDeployer</code>	Validates the Deployer configuration.
<code>waitForFlowRuntime</code>	Waits until the pipeline specified by the flow runtime ID is completed or the timeout expires.

Plugin Management

Commands	Description
<code>createPlugin</code>	Creates a plugin from an existing project.
<code>deletePlugin</code>	Deletes an existing plugin object without deleting the associated project or files.
<code>getPlugin</code>	Retrieves an installed plugin.
<code>getPlugins</code>	Retrieves all installed plugins.
<code>installPlugin</code>	Installs a plugin from a JAR file. Extracts the JAR contents on the server and creates a project and a plugin.
<code>modifyPlugin</code>	Modifies a plugin.
<code>promotePlugin</code>	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 <code>true</code> on one version sets the flag to <code>false</code> on all other plugins with the same key. The promoted version is the one resolved by an indirect reference of the form <code>\$/plugins/<key></code> or a plugin name argument without a specified version.
<code>uninstallPlugin</code>	Uninstalls a plugin, deleting the associated project and any installed files.

Procedure Management

Commands	Description
<code>createProcedure</code>	Creates a new procedure for an existing project.
<code>createStep</code>	Creates a new procedure step.
<code>deleteProcedure</code>	Deletes a procedure, including all steps.

Commands	Description
<code>deleteStep</code>	Deletes a step from a procedure.
<code>getProcedure</code>	Finds a procedure by its name.
<code>getProcedures</code>	Retrieves all procedures in a project.
<code>getStep</code>	Retrieves a step from a procedure.
<code>getSteps</code>	Retrieves all steps in a procedure.
<code>modifyProcedure</code>	Modifies an existing procedure.
<code>modifyStep</code>	Modifies an existing step.
<code>moveStep</code>	Moves a step within a procedure.

Process

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

Process Dependency Management

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

Process Step Management

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

Project Management

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

Property Management

Commands	Description
<code>createProperty</code>	Creates a regular string or nested property sheet using a combination of property path and context.
<code>deleteProperty</code>	Deletes a property from a property sheet.
<code>expandString</code>	Expands property references in a string, in the current context.
<code>getProperties</code>	Retrieves all properties associated with an object.

Commands	Description
<code>getProperty</code>	Retrieves the specified property value.
<code>incrementProperty</code>	Atomically increments the specified property value by the <code>incrementBy</code> amount. If the property does not exist, it will be created with an initial value of the <code>incrementBy</code> amount.
<code>modifyProperty</code>	Modifies a regular string or nested property sheet using a combination of property path and context.
<code>setProperty</code>	Sets the value for the specified property.

Resource Management

Commands	Description
<code>addResourcesToPool</code>	Adds resources to a specified resource pool.
<code>addResourceToEnvironmentTier</code>	Adds a resource to the specified environment tier.
<code>createResource</code>	Creates a new resource.
<code>createResourcePool</code>	Creates a pool container for resource.
<code>deleteResource</code>	Deletes a resource.
<code>deleteResourcePool</code>	Deletes a resource pool.
<code>getResource</code>	Retrieves a resource by its name.
<code>getResources</code>	Retrieves all resources.
<code>getResourcesInEnvironmentTier</code>	Retrieves the list of resources in an environment tier.
<code>getResourcesInPool</code>	Retrieves a list of resources in a pool.
<code>getResourcePool</code>	Retrieves a specified resource pool by name.
<code>getResourcePools</code>	Retrieves a list of resource pools.
<code>getResourceUsage</code>	Retrieves resource usage information.
<code>modifyResource</code>	Modifies an existing resource.
<code>modifyResourcePool</code>	Modifies an existing resource pool.

Commands	Description
<code>pingAllResources</code>	Pings all resources.
<code>pingResource</code>	Pings one resource.
<code>removeResourceFromEnvironmentTier</code>	Removes a resource from the specified environment tier.
<code>removeResourcesFromPool</code>	Removes resources from a specified resource pool.
<code>signCertificate</code>	Signs an agent certificate.

Schedule Management

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

Server Management

Commands	Description
<code>deleteLicense</code>	Deletes a license.
<code>getAdminLicense</code>	Retrieves the admin license, which can be used when all concurrent user licenses are in use.
<code>getCertificates</code>	Returns the certificates in the trust chain for the server.
<code>getLicense</code>	Retrieves information for one license.
<code>getLicenses</code>	Retrieves all license data.
<code>getLicenseUsage</code>	Retrieves the current license usage.

Commands	Description
<code>getServerInfo</code>	Retrieves information about server ports and message delivery.
<code>getServerStatus</code>	Returns the status of the server.
<code>getVersions</code>	Retrieves server version information.
<code>importLicenseData</code>	Imports one or more licenses.
<code>logMessage</code>	Enters a message in the server log.
<code>setLogLevel</code>	Changes the log level of a logger.
<code>shutdownServer</code>	Shuts down the ElectricFlow server.
<code>tunePerformance</code>	Adjusts how the server is performing.

Snapshot Management

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

Tier Map Management

Commands	Description
<code>createTierMap</code>	Creates a tier map for an application.
<code>deleteTierMap</code>	Deletes a tier map from an application.

Commands	Description
<code>deleteTierMapping</code>	Deletes a tier mapping from a tier map.
<code>getTierMaps</code>	Retrieves all tier maps that are used by an application.
<code>modifyTierMap</code>	Modifies an existing tier map.

User/Group Management

Commands	Description
<code>login</code>	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.
<code>logout</code>	Logs out of the client session.
<code>addUsersToGroup</code>	Adds ones or more specified users to a particular group.
<code>createGroup</code>	Creates a new local group of users.
<code>createUser</code>	Creates a new local user.
<code>deleteGroup</code>	Deletes a local group.
<code>deleteUser</code>	Deletes a local user.
<code>getGroup</code>	Retrieves a group by its name.
<code>getGroups</code>	Retrieves all groups.
<code>getUser</code>	Retrieves a user by name.
<code>getUsers</code>	Retrieves all users.
<code>login</code>	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.
<code>logout</code>	Logs out of the client session.
<code>modifyGroup</code>	Modifies an existing group.
<code>modifyUser</code>	Modifies an existing user.
<code>removeUsersFromGroup</code>	Removes one or more users from a particular group.

Workflow Definition Management

Commands	Description
<code>createStateDefinition</code>	Creates a new state definition for a workflow definition.
<code>createTransitionDefinition</code>	Creates a new transition definition for a workflow definition.
<code>createWorkflowDefinition</code>	Creates a new workflow definition for a project.
<code>deleteStateDefinition</code>	Deletes a state definition.
<code>deleteTransitionDefinition</code>	Deletes a transition definition.
<code>deleteWorkflowDefinition</code>	Deletes a workflow definition, including all state and transition definitions.
<code>getStateDefinition</code>	Finds a state definition by name.
<code>getStateDefinitions</code>	Retrieves all state definitions in a workflow definition.
<code>getTransitionDefinition</code>	Finds a transition definition by name.
<code>getTransitionDefinitions</code>	Retrieves all transition definitions in a workflow definition.
<code>getWorkflowDefinition</code>	Finds a workflow definition by name.
<code>getWorkflowDefinitions</code>	Retrieves all workflow definitions in a project.
<code>modifyStateDefinition</code>	Modifies an existing state definition.
<code>modifyTransitionDefinition</code>	Modifies an existing transition definition.
<code>modifyWorkflowDefinition</code>	Modifies an existing workflow definition.
<code>moveStateDefinition</code>	Moves a state definition within a workflow definition.
<code>moveTransitionDefinition</code>	Moves a transition definition within a workflow definition.

Workflow Management

Commands	Description
<code>completeWorkflow</code>	Marks a workflow as complete, which means transitions are no longer evaluated.

Commands	Description
<code>deleteWorkflow</code>	Deletes a workflow, including all states and transitions.
<code>getState</code>	Finds a state by name.
<code>getStates</code>	Retrieves all states in a workflow.
<code>getTransition</code>	Finds a transition by name.
<code>getTransitions</code>	Retrieves all transitions in a workflow.
<code>getWorkflow</code>	Finds a workflow by name.
<code>getWorkflows</code>	Retrieves all workflow instances in a project.
<code>runWorkflow</code>	Runs the specified workflow definition, returns the workflow name.
<code>transitionWorkflow</code>	Manually transition from a workflow active state.

Workspace Management

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

Miscellaneous

Commands	Description
<code>acquireNamedLock</code>	Retrieves the named lock.
<code>changeOwner</code>	Changes the owner of an object.

Commands	Description
<code>clone</code>	Makes a copy of an existing ElectricFlow project, procedure, step, schedule, resource, directory provider, email configuration, or email notifier.
<code>countObjects</code>	Returns the count of objects specified by the provided filter.
<code>deleteObjects</code>	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.
<code>dumpHeap</code>	Captures a Java heap dump.
<code>dumpStatistics</code>	Prints (emits) internal timing statistics.
<code>evalDsl</code>	Evaluates and runs an ElectricFlow domain-specific language (DSL) script.
<code>evalScript</code>	Evaluates a script in a given context. This API is similar to <code>expandString</code> except that it evaluates the <code>value</code> 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 <code>value</code> element of the response.
<code>export</code>	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.
<code>findObjects</code>	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 <code>findObjects</code> command.
<code>finishCommand</code>	The agent uses this command to indicate that a command has been run.
<code>generateDsl</code>	Generates domain-specific language (DSL) script for an existing object.
<code>getObjects</code>	Retrieves the full object based on IDs returned by <code>findObjects</code> . All requested objects must be of the same <code>objectType</code> . See <code>findObjects</code> for a list of object types.

Commands	Description
<code>graphStateMachine</code>	Generates a graph element with a stateMachine DOT graph as CDATA content.
<code>import</code>	Imports data from an XML export file.
<code>logStatistic</code>	Prints (emits) a statistics value to StatsD.
<code>releaseNamedLock</code>	Releases the named lock that synchronizes the name of an object.

API Commands - ACL Management

[breakAclInheritance](#) on page 76

[checkAccess](#) on page 81

[createAclEntry](#) on page 86

[deleteAclEntry](#) on page 92

[getAccess](#) on page 97

[getAclEntry](#) on page 102

[modifyAclEntry](#) on page 107

[restoreAclInheritance](#) on page 112

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:	
<code>applicationName</code>	The name of the application. Argument type: String
<code>applicationTierName</code>	The name of the application tier. Argument type: String

Arguments	Descriptions
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
credentialName	The name of the credential that can be one of these formats: <ul style="list-style-type: none"> • 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

Arguments	Descriptions
<code>flowRuntimeName</code>	The name of the flow runtime. Argument type: String
<code>flowStateName</code>	The name of the flow state. Argument type: String
<code>flowTransitionName</code>	The name of the flow transition. Argument type: String
<code>gatewayName</code>	The name of the gateway. Argument type: String
<code>groupName</code>	The full name of the group. For Active Directory and LDAP, the full name if the full domain name. Argument type: String
<code>jobId</code>	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
<code>jobStepId</code>	The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>notifierName</code>	The name of the email notifier. Argument type: String
<code>objectId</code>	This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . 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
<code>path</code>	The property path. Argument type: String
<code>pipelineName</code>	The name of the pipeline. Argument type: String
<code>pluginName</code>	The plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin. Argument type: String

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure or a path to a procedure that includes the name. If you use this argument, you must also use <code>projectName</code> . Argument type: String
<code>processName</code>	The name of the process if the container is a process or process step. Argument type: String
<code>processStepName</code>	The name of the process step if the container is a process step. Argument type: String
<code>projectName</code>	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
<code>propertySheetId</code>	The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
<code>releaseName</code>	The name of the Release. Argument type: String
<code>repositoryName</code>	The name of the repository used for artifact management. Argument type: String
<code>resourceName</code>	The name of the resource. Argument type: String
<code>resourcePoolName</code>	The name of the resource pool. Argument type: String
<code>resourceTemplateName</code>	The name of the resource template.
<code>scheduleName</code>	The name of a schedule, which can be a path to the schedule. If you use this argument, you must also use <code>projectName</code> . Argument type: String
<code>snapshotName</code>	The name of the snapshot. Argument type: String

Arguments	Descriptions
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
stepName	The name of the step, which can be a path to the step. If you use this argument, you must also use <code>projectName</code> and <code>procedureName</code> . Argument type: String
systemObjectName	The name of the system object. System objects names include: <code>admin artifactVersions directory emailConfigs log plugins server session workspaces</code> . 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 <code>user@domain</code> . 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

Arguments	Descriptions
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: \$cmdr->breakAclInheritance({<optionals>});

Example

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

ectool

syntax: ectool breakAclInheritance [optionals]

Example

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

[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

Arguments	Descriptions
artifactVersionName	<p>The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "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.</p> <p>Argument type: String</p>
componentName	<p>The name of the component.</p> <p>Argument type: String</p>
configName	<p>The name of the email configuration.</p> <p>Argument type: String</p>
credentialName	<p>The name of the credential container of the property sheet that owns the property.</p> <p>Specify <code>credentialName</code> using one of two forms:</p> <ul style="list-style-type: none"> • 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. <p>Argument type: String</p>
environmentName	<p>The name of the environment that must be unique among all projects.</p> <p>Argument type: String</p>
environmentTemplateName	<p>The name of the environment template.</p> <p>Argument type: String</p>
environmentTemplateTierName	<p>The name of the environment template tier.</p> <p>Argument type: String</p>
environmentTierName	<p>The name of the environment tier.</p> <p>Argument type: String</p>
flowName	<p>The name of the flow that must be unique within the project.</p> <p>Argument Type: String</p>

Arguments	Descriptions
<code>flowRuntimeName</code>	The name of the flow runtime that must be unique within the flow. Argument Type: String
<code>flowRuntimeStateName</code>	The name of the flow run-time state. Argument Type: String
<code>flowStateName</code>	Name of the flow state that must be unique within the flow. Argument Type: String
<code>flowTransitionName</code>	Name of the flow transition that must be unique within the flow state. Argument Type: String
<code>gatewayName</code>	The name of the gateway. Argument type: String
<code>groupName</code>	The full name of the group. For Active Directory and LDAP, this is a full domain name. Argument type: String
<code>jobId</code>	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
<code>jobStepId</code>	The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>notifierName</code>	The name of the email notifier. Argument type: String
<code>objectId</code>	The object identifier returned by <code>findObjects</code> and <code>getObjects</code> . 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
<code>path</code>	Property path string. Argument type: String
<code>pipelineName</code>	The name of the pipeline. Argument type: String

Arguments	Descriptions
<code>pluginName</code>	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
<code>procedureName</code>	The name of the procedure. It can be be a path to the procedure. When using this procedure, you must also use <code>projectName</code> . Argument type: String
<code>processName</code>	The name of the process. Argument type: String
<code>processStepName</code>	The name of the process step. Argument type: String
<code>projectName</code>	The name of the project that must be unique 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
<code>propertySheetId</code>	The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
<code>releaseName</code>	The name of the Release that owns the property. Argument type: String
<code>repositoryName</code>	The name of the repository for artifact management. Argument type: String
<code>resourceName</code>	The name of the resource. Argument type: String
<code>resourcePoolName</code>	The name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	The name of the resource template. Argument type: String
<code>scheduleName</code>	The name of the schedule, which can be the path to the schedule. When using this argument, you must also enter <code>projectName</code> . Argument type: String

Arguments	Descriptions
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
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 <code>projectName</code> and <code>procedureName</code> . Argument type: String
systemObjectName	System object names include: <code>admin directory licensing log plugins priority projects </code> . 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 <code>user@domain</code> . 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

Arguments	Descriptions
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: `$cmdr->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 <code>user</code> or <code>group</code> . Argument type: <code>PrincipalType</code>
principalName	This is either a user or a group name. Argument type: <code>PrincipalName</code>
Locators:	
applicationName	The name of the application that must be unique among all projects. Argument type: <code>String</code>

Arguments	Descriptions
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 <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> 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 (<code>emailConfig</code>). Argument type: String
credentialName	The name of the credential specified in one of these formats: <ul style="list-style-type: none"> • relative(for example, "<code>cred1</code>")–The credential is assumed to be in the project that contains the requested target object. • absolute(for example, "<code>projects/BuildProject/credentials/cred1</code>")–The credential can be from any specified project, regardless of the project for the target object. When using this argument, you must also enter <code>projectName</code> . 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

Arguments	Descriptions
environmentTemplateTierName	The name of the environment template tier. Argument type: String
environmentTierName	The name of the environment tier. Argument type: String
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

Arguments	Descriptions
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
objectId	The object identifier returned by <code>findObjects</code> and <code>getObjects</code> . 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 <code>projectName</code> . 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

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

Arguments	Descriptions
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. 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: \$cmdr->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
<code>principalType</code>	A user or a group: <user group>. The default is to user. Argument type: PrincipalType
<code>principalName</code>	The name of the user or the group. Argument type: PrincipalName
<code>applicationName</code>	(Optional) The name of the application that must be unique among all projects. Argument type: String
<code>applicationTierName</code>	(Optional) The name of the application tier. Argument type: String
<code>artifactName</code>	(Optional) The name of the artifact. Argument type: String
<code>artifactVersionName</code>	(Optional) The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the artifactVersion in question. This name is parsed and interpreted as "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
<code>componentName</code>	(Optional) The name of the component. Argument type: String
<code>configName</code>	(Optional) The name of the email configuration. Argument type: String

Arguments	Descriptions
<code>credentialName</code>	<p>(Optional) The name of the credential specified in one of these formats:</p> <ul style="list-style-type: none"> • relative(for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the requested target object. • absolute(for example, "<i>/projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the project for the target object. <p>Argument type: String</p>
<code>environmentName</code>	<p>(Optional) The name of the environment that must be unique among all projects.</p> <p>Argument type: String</p>
<code>environmentTemplateName</code>	<p>(Optional) The name of the environment template that must be unique among all projects.</p> <p>Argument type: String</p>
<code>environmentTemplateTierName</code>	<p>(Optional) Name of the environment template tier that must be unique among all tiers for the environment template.</p> <p>Argument Type: String</p>
<code>environmentTierName</code>	<p>(Optional) The name of the environment tier.</p> <p>Argument type: String</p>
<code>flowName</code>	<p>(Optional) The name of the flow.</p> <p>Argument type: String</p>
<code>flowRuntimeName</code>	<p>(Optional) Name of the flow runtime.</p> <p>Argument Type: String</p>
<code>flowRuntimeStateName</code>	<p>(Optional) Name of the flow state.</p> <p>Argument Type: String</p>
<code>flowStateName</code>	<p>(Optional) The name of the flow state.</p> <p>Argument type: String</p>
<code>flowTransitionName</code>	<p>(Optional) The name of the flow transition.</p> <p>Argument type: String</p>

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

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

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

Positional arguments

`principalType, principalName`

Response

None or a status OK message.

ec-perl

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

Example

```
$cmdr->deleteAclEntry('user', 'j smith', {projectName => '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 <code>artifactVersionName</code> 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	<p>(Optional) The name of the credential specified in one of these formats:</p> <ul style="list-style-type: none"> • 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. <p>Argument type: String</p>
emulateRestoreInheritance	<p>(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.</p> <p><i><Boolean flag - 0 1 true false></i> If set to true or 1, this argument returns ACL information to what it would be if inheritance were restored on this object.</p> <p>Argument type: Boolean</p>
environmentName	<p>(Optional) The name of the environment that must be unique among all projects.</p> <p>Argument type: String</p>
environmentTemplateName	<p>(Optional) Name of the environment template.</p> <p>Argument type: String</p>
environmentTemplateTierName	<p>(Optional) Name of the environment template tier.</p> <p>Argument type: String</p>
environmentTierName	<p>(Optional) The name of the environment tier.</p> <p>Argument type: String</p>
flowName	<p>(Optional) The name of the flow.</p> <p>Argument type: String</p>
flowRuntimeName	<p>(Optional) Name of the flow runtime.</p> <p>Argument Type: String</p>
flowRuntimeStateName	<p>(Optional) Name of the flow state.</p> <p>Argument Type: String</p>

Arguments	Descriptions
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>flowTransitionName</code>	(Optional) The name of the flow transition. Argument type: String
<code>gatewayName</code>	(Optional) The name of the gateway. Argument type: String
<code>groupName</code>	(Optional) The name of the group. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String
<code>notifierName</code>	(Optional) The name of the email notifier with the ACL. Argument type: String
<code>objectId</code>	(Optional) An object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(Optional) Property path. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(Optional) The name of the plugin with the ACL. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure with the ACL. When using this argument, you must also enter <code>projectName</code> . 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 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.
stateName	(Optional) The name of the state.

Arguments	Descriptions
<code>stepName</code>	(Optional) The name of the step containing the ACL. When using this argument, you must also enter <code>projectName</code> Argument type: String
<code>systemObjectName</code>	(Optional) System objects include: <code>admin artifactVersions directory emailConfigs log plugins server session workspaces</code> Argument type: SystemObjectName
<code>taskName</code>	(Optional) The name of the task. Argument type: String
<code>transitionDefinitionName</code>	(Optional) The name of the transition definition. Argument type: String
<code>transitionName</code>	(Optional) The name of the transition. Argument type: String
<code>userName</code>	(Optional) The name of the user with the ACL. Argument type: String
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workflowName</code>	(Optional) The name of the workflow. Argument type: String
<code>workspaceName</code>	(Optional) The name of the workspace with the ACL. Argument type: String
<code>zoneName</code>	(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: `$cmdr->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
<code>principalType</code>	Type of principal for this ACE: <code>user</code> or <code>group</code> . Argument type: <code>PrincipalType</code>
<code>principalName</code>	Name of the user or group for the ACE. Argument type: <code>PrincipalName</code>
<code>applicationName</code>	(Optional) The name of the application that must be unique among all projects. Argument type: <code>String</code>
<code>applicationTierName</code>	(Optional) The name of the application tier. Argument type: <code>String</code>
<code>artifactName</code>	(Optional) The name of the artifact. Argument type: <code>String</code>
<code>artifactVersionName</code>	(Optional) The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name. The ElectricFlow server interprets the name form correctly. Argument type: <code>String</code>

Arguments	Descriptions
<code>componentName</code>	(Optional) The name of the component. Argument type: String
<code>configName</code>	(Optional) The name of the email configuration. Argument type: String
<code>credentialName</code>	(Optional) The name of the credential specified in one of these formats: <ul style="list-style-type: none"> • relative(for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the requested target object. • absolute(for example, "<i>/projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the project for the target object. Argument type: String
<code>environmentName</code>	(Optional) The name of the environment that must be unique among all projects. Argument type: String
<code>environmentTemplateName</code>	(Optional) Name of the environment template. Argument type: String
<code>environmentTemplateTierName</code>	(Optional) Name of the environment template tier. Argument type: String
<code>environmentTierName</code>	(Optional) Name of the environment tier. Argument type: String
<code>flowName</code>	Name of the flow that must be unique within the project. Argument Type: String
<code>flowRuntimeName</code>	(Optional) Name of the flow runtime. Argument Type: String
<code>flowRuntimeStateName</code>	(Optional) Name of the flow state. Argument Type: String
<code>flowStateName</code>	Name of the flow state that must be unique within the flow. Argument Type: String

Arguments	Descriptions
<code>flowTransitionName</code>	Name of the flow transition that must be unique within the flow state. Argument Type: String
<code>gatewayName</code>	(Optional) The name of the gateway. Argument type: String
<code>groupName</code>	(Optional) The name of the group. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: String
<code>objectId</code>	(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(Optional) The property path. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(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
<code>procedureName</code>	(Optional) The name of the procedure with the ACL. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process. Argument type: String

Arguments	Descriptions
<code>processStepName</code>	(Optional) The name of the process step. Argument type: String
<code>projectName</code>	(Optional) The name of the project. Argument type: String
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: String
<code>releaseName</code>	(Optional) The name of the Release. Argument type: String
<code>repositoryName</code>	(Optional) The name of the repository for artifact management. Argument type: UUID
<code>resourceName</code>	(Optional) The name of the resource. Argument type: String
<code>resourcePoolName</code>	(Optional) The name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	(Optional) The name of the resource template. Argument type: String
<code>scheduleName</code>	(Optional) The name of a schedule. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>snapshotName</code>	(Optional) The name of a snapshot. Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(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 <code>projectName</code> and <code>procedureName</code> . Argument type: String
systemObjectName	(Optional) System objects include: <code>admin artifactVersions directory emailConfigs log plugins server session workspaces</code> 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
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
<code>principalType</code>	Type of principal for this access control entry: <code>user</code> or <code>group</code> . Argument type: <code>PrincipalType</code>
<code>principalName</code>	Name of the user or group for this access control entry. Argument type: <code>PrincipalName</code>
<code>applicationName</code>	(Optional) The name of the application that must be unique among all projects. Argument type: <code>String</code>
<code>applicationTierName</code>	(Optional) The name of the application tier. Argument type: <code>String</code>
<code>artifactName</code>	(Optional) The name of the artifact. Argument type: <code>String</code>
<code>artifactVersionName</code>	(Optional) The name of the artifact version. An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as " <code>groupId:artifactKey:version</code> " and the object is searched either way you specify its name. The ElectricFlow server interprets the name form correctly. Argument type: <code>String</code>

Arguments	Descriptions
<code>changePermissionsPrivilege</code>	(Optional) <code><allow deny></code> –Determines whether the principal can modify access control for the object. Argument type: Access
<code>componentName</code>	(Optional) The name of the component. Argument type: String
<code>configName</code>	(Optional) The name of the email configuration. Argument type: String
<code>credentialName</code>	(Optional) The name of the credential specified in one of these formats: <ul style="list-style-type: none"> • relative(for example, <i>"cred1"</i>)–The credential is assumed to be in the project that contains the requested target object. • absolute(for example, <i>"projects/BuildProject/credentials/cred1"</i>)–The credential can be from any specified project, regardless of the project for the target object. Argument type: String
<code>environmentName</code>	(Optional) The name of the environment that must be unique among all projects. Argument type: String
<code>environmentTemplateName</code>	(Optional) The name of the environment template. Argument type: String
<code>environmentTemplateTierName</code>	(Optional) The name of the environment template tier. Argument type: String
<code>environmentTierName</code>	(Optional) The name of the environment tier. Argument type: String
<code>executePrivilege</code>	(Optional) <code><allow deny></code> –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
<code>flowName</code>	(Optional) The name of the flow that must be unique within the project. Argument Type: String

Arguments	Descriptions
<code>flowRuntimeName</code>	(Optional) The name of the flow runtime. Argument Type: String
<code>flowRuntimeStateName</code>	(Optional) The name of the flow state. Argument Type: String
<code>flowStateName</code>	(Optional) The name of the flow state that must be unique within the flow. Argument Type: String
<code>flowTransitionName</code>	(Optional) The name of the flow transition that must be unique within the flow state. Argument Type: String
<code>gatewayName</code>	(Optional) The name of the gateway. Argument type: String
<code>groupName</code>	(Optional) The name of the group with the ACE. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>modifyPrivilege</code>	(Optional) <code><allow deny></code> –Determines whether the principal can change the contents of the object. Argument type: Access
<code>notifierName</code>	(Optional) The name of the email notifier with the ACE. Argument type: String
<code>objectId</code>	(Optional) The object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(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 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 <code>projectName</code> . 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
readPrivilege	(Optional) <code><allow deny></code> –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

Arguments	Descriptions
<code>scheduleName</code>	(Optional) The name of the schedule with the ACE. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>snapshotName</code>	(Optional) The name of a snapshot. Argument type: String
<code>stageName</code>	(Optional) The name of the stage. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) The name of the step with the ACE. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>systemObjectName</code>	(Optional) System object names include: <code>admin artifacts directory emailConfigs forceAbort licensing log plugins priority projects repositories resources server session test workspaces zonesAndGateways</code> Argument type: <code>SystemObjectName</code>
<code>taskName</code>	(Optional) The name of the task in a stage in a pipeline. Argument type: String
<code>transitionDefinitionName</code>	(Optional) The name of the transition definition. Argument type: String
<code>transitionName</code>	(Optional) The name of the transition. Argument type: String
<code>userName</code>	(Optional) The user name containing the ACE. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String

Arguments	Descriptions
<code>workflowName</code>	The name of the workflow. Argument type: String
<code>workspaceName</code>	The name of the workspace containing the ACE. Argument type: String
<code>zoneName</code>	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
<code>applicationName</code>	(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. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as " <code>groupId:artifactKey:version</code> " 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: <ul style="list-style-type: none">• relative(for example, "<code>cred1</code>")—The credential is assumed to be in the project that contains the requested target object.• absolute(for example, "<code>/projects/BuildProject/credentials/cred1</code>")—The credential can be from any specified project, regardless of the project for the target object. When using this argument, you must also enter <code>projectName</code> . 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

Arguments	Descriptions
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
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

Arguments	Descriptions
<code>objectId</code>	(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> .
<code>path</code>	(Optional) Property path. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(Optional) The name of the plugin with the ACL inheritance that you want to restore. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure with the ACL inheritance that you want to restore. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step. Argument type: String
<code>projectName</code>	(Optional) The name of the project with the ACL inheritance that you want to restore. Argument type: String
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
<code>releaseName</code>	(Optional) The name of the Release. Argument type: String
<code>repositoryName</code>	(Optional) The name of the repository for artifact management. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource whose ACL inheritance you want to restore. Argument type: String

Arguments	Descriptions
<code>resourcePoolName</code>	(Optional) The name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	(Optional) Name of the resource template. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule with the ACL inheritance that you want to restore. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>snapshotName</code>	(Optional) The name of a snapshot. Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) The name of the step with the ACL inheritance that you want to restore. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>systemObjectName</code>	(Optional) The name of the system object whose ACL inheritance you want to restore. System objects include: <code>admin artifactVersions directory emailConfigs log plugins server session workspaces</code> Argument type: <code>SystemObjectName</code>
<code>taskName</code>	(Optional) The name of the task. Argument type: String
<code>transitionDefinitionName</code>	(Optional) The name of the transition definition. Argument type: String
<code>transitionName</code>	(Optional) The name of the transition. Argument type: String

Arguments	Descriptions
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
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](#) on page 118

[deleteApplication](#) on page 118

[getApplication](#) on page 119

[getApplications](#) on page 120

[modifyApplication](#) on page 121

createApplication

Creates a new application for a project.

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

Arguments	Descriptions
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application that must be unique among all projects. Argument Type: String
<code>description</code>	(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-73f1-11e3-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) <i><Boolean flag - 0 1 true false></i> 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 Summaries"});
```

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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application that must be unique within the project. Argument Type: String
<code>description</code>	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
<code>newName</code>	(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", description => "Pet store website"});
```

ectool

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

Example

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

[Back to Top](#)

API Commands - Application Tier

[createApplicationTier](#) on page 122

[deleteApplicationTier](#) on page 123

[getApplicationTier](#) on page 124

[getApplicationTiers](#) on page 125

[getApplicationTiersInComponent](#) on page 126

[modifyApplicationTier](#) on page 127

createApplicationTier

Creates a new application tier in the application.

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

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

Arguments	Descriptions
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 setup"
```

[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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application that must be unique among all projects. Argument Type: String
<code>applicationTierName</code>	Name of the tier that must be unique within the application. Argument Type: String
<code>applicationEntityRevisionId</code>	(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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application that must be unique among all projects. Argument Type: String
<code>applicationEntityRevisionId</code>	(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 => "4fa765dd-73f1-11e3-b67e-b0a420524153"});
```

ectool

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

Example

```
ectool getApplicationTiers "Default" "Deploy" --applicationEntityRevisionId 4fa765dd-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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>componentName</code>	Name of the component. Argument Type: String
<code>applicationEntityRevisionId</code>	(Optional) The revision ID of the versioned object. Argument type: UUID
<code>applicationName</code>	(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 => "Snapshot"});
```


ectool

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

Example

```
ectool getApplicationTiersInComponent "Default" "WAR file" --applicationName "Snapshot"
```

[Back to Top](#)

modifyApplicationTier

Modifies an existing tier in the application.

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

Arguments	Descriptions
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application that must be unique within the project. Argument Type: String
<code>applicationTierName</code>	Name of the tier that must be unique within the application. Argument Type: String
<code>description</code>	(Optional) Comment text describing this object, which is not interpreted by ElectricFlow. Argument Type: String
<code>newName</code>	(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=> "Deploy README", description=> "Revised README file"});
```

ectool

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

Example

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

[Back to Top](#)

API Commands - Artifact Management

[addDependentsToArtifactVersion](#) on page 129

[cleanupArtifactCache](#) on page 130

[cleanupRepository](#) on page 131

[createArtifact](#) on page 132

[createArtifactVersion](#) on page 133

[createRepository](#) on page 134

[deleteArtifact](#) on page 135

[deleteArtifactVersion](#) on page 136

[deleteRepository](#) on page 137

[findArtifactVersions](#) on page 138

[getArtifact](#) on page 141

[getArtifacts](#) on page 142

[getArtifactVersion](#) on page 142

[getArtifactVersions](#) on page 144

[getManifest](#) on page 144

[getRepositories](#) on page 145

[getRepository](#) on page 146

[getRetrievedArtifacts](#) on page 146

[modifyArtifact](#) on page 147

[modifyArtifactVersion](#) on page 148

[modifyRepository](#) on page 150

[moveRepository](#) on page 151

[publishArtifactVersion](#) on page 152

[removeDependentsFromArtifactVersion](#) on page 155

[resolveRoute](#) on page 157

[retrieveArtifactVersions](#) on page 157

[updateArtifactVersion](#) on page 162

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
<code>artifactVersionName</code>	<p>The name of the artifact version.</p> <p>An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name. The ElectricFlow server interprets the name form correctly.</p> <p>Argument type: String</p>
<code>dependentArtifactVersions</code>	<p>(Optional) One or more artifact version queries. The most current match of each query is retrieved when the primary artifact is retrieved.</p> <p>Dependent artifact version query strings are in this form: <code><groupId>:<artifactKey>:<versionRange></code> (versionRange is optional).</p> <p>The version range syntax is standard number interval notation. <code>()</code> marks exclusive ranges and <code>[]</code> marks inclusive ranges.</p> <p>Argument type: Collection</p>

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
<code>cacheDirectory</code>	The directory where stale artifact versions are stored. Argument type: String
<code>force</code>	<i><Boolean flag - 0 1 true false></i> 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
<code>backingStoreDirectory</code>	The repository directory where artifact versions are stored. Argument type: String
<code>force</code>	<i><Boolean flag - 0 1 true false></i> 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
<code>groupId</code>	A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String
<code>artifactKey</code>	A user-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String
<code>artifactVersionNameTemplate</code>	(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
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> 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
<code>version</code>	The version component of the GAV (<code>groupId/artifactVersionId/version</code>) coordinates. Argument type: String
<code>artifactKey</code>	(Optional) A user-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String
<code>artifactName</code>	(Optional) The name of the artifact. Argument type: String
<code>dependentArtifactVersions</code>	(Optional) The set of artifact versions on which this <code>artifactVersion</code> depends. Argument type: Collection
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String
<code>groupId</code>	(Optional) The <code>groupId</code> component of the GAV (<code>groupId/artifactVersionId/version</code>) coordinates. This field is limited to alphanumeric characters, spaces, spaces, underscores, hyphens, and periods. Argument type: String
<code>jobStepId</code>	(Optional) The unique identifier for the job step that is used to make a project association. Argument type: UUID

Arguments	Descriptions
repositoryName	(Optional) The name of the artifact repository. Argument type: String

Positional arguments

version

Response

Returns an [artifactVersion](#) on page 739 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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String

Arguments	Descriptions
repositoryDisabled	(Optional) <i><Boolean flag -0 1 true false></i> – 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.eccloud.com:8200"});
```

ectool

syntax: ectool createRepository <repositoryName> [optionals]

Example

```
ectool createRepository myRepos --repositoryDisabled "true" --url
    "https://test.eccloud.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	<p>The name of the artifact version.</p> <p>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.</p> <p>Argument type: String</p>

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

Arguments	Descriptions
filters	<p>A list of zero or more filter criteria definitions used to define objects to find.</p> <p>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.</p> <p>Two types of filters:</p> <p>Property filters are used to select objects based on the value of the object's intrinsic or custom property.</p> <p>Boolean filters ("and", "or", "not") are used to combine one or more filters using Boolean logic.</p> <p>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.</p> <p>Property filter operators are:</p> <ul style="list-style-type: none"> 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) <p>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.</p> <p>Boolean operators are:</p> <ul style="list-style-type: none"> not (1 operand) and (2 or more operands)

Arguments	Descriptions
	<p>or (2 or more operands)</p> <p>Argument type: Collection</p>
groupId	<p>A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.</p> <p>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.</p> <p>This argument is required when there is no job step.</p> <p>Argument type: String</p>
includeDependents	<p><i><Boolean flag -0 1 true false></i></p> <p>Options are:</p> <ul style="list-style-type: none"> 0/false – dependent artifacts are not retrieved. 1/true – dependent artifacts are retrieved. <p>Argument type: Boolean</p>
jobStepId	<p>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.</p> <p>Argument type: UUID</p>
versionRange	<p>The range of versions to search. Version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.</p> <p>Argument type: String</p>

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 above.
$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
<code>artifactName</code>	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	<p>The name of the artifact version to retrieve.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly.</p> <p>Argument type: String</p>
includeRetrieverJobs	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>If set to 1 or true, this argument includes <code>jobId</code> and <code>jobName</code> in returned information. A retriever job is any job that has retrieved the artifact version.</p> <p>Argument type: Boolean</p>
includeRetrieverJobSteps	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>If set to 1 or true, this argument includes <code>jobId</code>, <code>jobName</code>, and <code>jobStepId</code> 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.</p> <p>Argument type: Boolean</p>
maxRetrievers	<p>(Optional) If one of the <code>includeRetriever*</code> options are specified, return at most "this many" of the most recent retrievers. Without this option, the ElectricFlow server will return all retrievers.</p> <p>Argument type: String</p>

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 => "true"});
```

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
<code>artifactName</code>	The name of the artifact for the versions to retrieve. Argument type: String
<code>retrieverJobId</code>	The job ID to use when retrieving an artifact. Argument type: UUID
<code>retrieverJobStepId</code>	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
<code>artifactVersionName</code>	<p>The name of the artifact version whose manifest you want to retrieve.</p> <p>Argument type: String</p>

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
<code>repositoryName</code>	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
<code>jobId</code>	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
<code>componentName</code>	(Optional) Name of the component. Argument type: String
<code>resourceName</code>	(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
<code>artifactName</code>	The name of the artifact to modify. Argument type: String

Arguments	Descriptions
<code>artifactVersionNameTemplate</code>	(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
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> 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 for SDK"
```

[Back to Top](#)

modifyArtifactVersion

Modifies an existing artifact version.

You must specify an `artifactVersionName`.

Arguments	Descriptions
artifactVersionName	<p>The name of the artifact version to modify.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "<code>groupId:artifactKey:version</code>" and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly.</p> <p>Argument type: String</p>
artifactVersionState	<p>(Optional) The state of the artifact version. <publishing available unavailable>.</p> <p>Argument type: ArtifactVersionState</p>
dependentArtifactVersions	<p>(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).</p> <p>Note: The absence of this argument does not clear or modify the dependent artifact version list for this artifact version.</p> <p>Argument type: Collection</p>
description	<p>(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>
 <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr> </p> <p>Argument type: String</p>
newName	<p>(Optional) New name for this artifact version.</p> <p>Argument type: String</p>
removeAllDependentArtifactVersions	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>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.</p> <p>Argument type: Boolean</p>
repositoryName	<p>(Optional) The name of the artifact repository.</p> <p>Argument type: String</p>

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> <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr> Argument type: String
newName	(Optional) New name of the repository. Argument type: String
repositoryDisabled	(Optional) <i><Boolean flag - 0 1 true false></i> 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 (<code>repositoryName</code>) to a place before the name specified by this option. If omitted <code>repositoryName</code> 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	<p>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: <code><groupId>:<artifactKey>:<versionRange></code> (<code>versionRange</code> is optional).</p> <p>The version range syntax is standard number interval notation. <code>()</code> marks exclusive ranges and <code>[]</code> marks inclusive ranges.</p> <p>Argument type: Collection</p>
description	<p>A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
excludePatterns	<p>Semi-colon delimited list of file-path patterns indicating which files/directories under "<code>fromDirectory</code>" to exclude when publishing an artifact version. Defaults to "empty," which means no files are excluded. See more information on "pattern syntax" below.</p> <p>Argument type: Collection</p>
followSymlinks	<p><Boolean flag - 0 1 true false> Default is "true".</p> <p>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.</p> <p>Argument type: Boolean</p>
fromDirectory	<p>The directory containing files to publish as the artifact version. A subset of files can be published based on <code>includePatterns</code> and <code>excludePatterns</code>.</p> <p>Argument type: String</p>
groupId	<p>A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>includePatterns</code>	<p>Semi-colon delimited list of file-path patterns indicating which files/directories under "<code>fromDirectory</code>" 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.</p> <p>Argument type: Collection</p>
<code>repositoryName</code>	<p>The name of the artifact repository where you want to publish.</p> <p>Argument type: String</p>
<code>version</code>	<p>Unique identifier for the artifact version in the form: <code>major.minor.patch-qualifier-buildNumber</code> <code>major</code>, <code>minor</code>, <code>patch</code>, and <code>buildNumber</code> are integers and <code>qualifier</code> can contain any character except the following: <code>\:<> ?*/</code></p> <p>If a version argument is provided, but does not follow the above format, the version will be considered <code>0.0.0-<user-specified-version-arg>-0</code> implicitly. See the examples below.</p> <p>Argument type: String</p>

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	<p>The name of the artifact version from which you want to remove dependents.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name--the ElectricFlow server interprets either name form correctly.</p> <p>Argument type: String</p>
dependentArtifactVersions	<p>(Optional) One or more artifact version queries. The most current match of each query is retrieved when the primary artifact is retrieved.</p> <p>Dependent artifact version query strings are in this form: <code><groupId>:<artifactKey>:<versionRange></code> (versionRange is optional).</p> <p>The version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.</p> <p>Argument type: Collection</p>

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
<code>toRepositoryName</code>	Name of the artifact repository. Argument type: String
<code>fromAgentId</code>	(Optional) Identifier of the agent requesting the route to a destination agent or artifact repository. Argument type: Long
<code>fromResourceName</code>	(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
<code>artifactKey</code>	User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String
<code>artifactName</code>	The name of the artifact. Argument type: String
<code>artifactVersionName</code>	The name of the artifact version. Argument type: String
<code>cacheDirectory</code>	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	<p>A list of zero or more filter criteria definitions used to define objects to find.</p> <p>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.</p> <p>Two types of filters:</p> <ul style="list-style-type: none"> • 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. <p>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.</p> <p>Property filter operators are:</p> <ul style="list-style-type: none"> 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) <p>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.</p> <p>Boolean operators are:</p> <ul style="list-style-type: none"> not (1 operand) and (2 or more operands) or (2 or more operands)

Arguments	Descriptions
	Argument type: Collection
<code>groupId</code>	<p>A user-generated group name for this artifact.</p> <p>This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.</p> <p>Argument type: String</p>
<code>includeDependents</code>	<p><Boolean flag - 0 1 true false> The default is 1 or true.</p> <p>If the value is 1 or true:</p> <ul style="list-style-type: none"> • 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 <code>cacheDirectory</code> or if <code>toDirectory</code> is specified, under the <code>oDirectory/ec_dependent_artifacts</code> directory. <p>If the value is 0 or false, only the artifact is retrieved. The artifact version does not include the dependent artifacts.</p> <p>Argument type: Boolean</p>
<code>overwrite</code>	<p>Options are:</p> <ul style="list-style-type: none"> • <code>true</code> - deletes previous content in the directory and replaces the content with your new version. • <code>false</code> - (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. • <code>update</code> - 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. <p>Argument type: String</p>

Arguments	Descriptions
<code>repositoryNames</code>	<p>A space-separated list of artifact repository names.</p> <p>Retrieval is attempted from each specified repository in a specified order until it succeeds or all specified repositories have rejected the retrieval.</p> <p>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.</p> <p>Argument type: String</p>
<code>retryNumber</code>	<p>Number of retry attempts for the operation. The default is 1.</p> <p>The time between retry attempts is 20 seconds.</p> <p>Argument type: Integer</p>
<code>toDirectory</code>	<p>Used to retrieve an artifact version to a specific directory without imposing the structure of a cache directory.</p> <p>Specify the full path to the new directory.</p> <ul style="list-style-type: none"> • 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. <code>toDirectory</code> overrides <code>cacheDirectory</code> for downloads. <p>Argument type: String</p>
<code>versionRange</code>	<p>The range of versions to search.</p> <p>Version range syntax is standard number interval notation. <code>()</code> marks exclusive ranges and <code>[]</code> marks inclusive ranges.</p> <p>Argument type: String</p>

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

Arguments	Descriptions
excludePatterns	<p>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.</p> <p>Argument type: Collection</p>
followSymlinks	<p><Boolean flag - 0 1 true false> The default is 1 or true.</p> <p>If 1 or true, ElectricFlow follows symbolic links and record the target file contents with the symbolic link name in the artifact.</p> <p>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.</p> <p>Argument type: Boolean</p>
fromDirectory	<p>The directory containing files to publish as the artifact version. A subset of files can be published based on includePatterns and excludePatterns.</p> <p>Argument type: String</p>
groupId	<p>A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, spaces, underscores, hyphens, and periods.</p> <p>Argument type: String</p>
includePatterns	<p>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.</p> <p>Argument type: Collection</p>
newVersion	<p>Unique identifier for the new artifact version in the form: <code>major.minor.patch-qualifier-buildNumber</code> <code>major</code>, <code>minor</code>, <code>patch</code>, and <code>buildNumber</code> are integers and <code>qualifier</code> can contain any character except the following: <code>\:<> ?*/</code></p> <p>If a version argument is provided, but does not follow the above format, the version will be considered <code>0.0.0-<user-specified-version-arg>-0</code> implicitly.</p> <p>See examples below.</p> <p>Argument type: String</p>

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- <code><user-specified-version-arg>-0</code> 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

[getDeploymentHistoryItems](#) on page 165

[getEntityChange](#) on page 166

[getEntityChangeDetails](#) on page 167

[pruneChangeHistory](#) on page 168

[revert](#) on page 169

[searchEntityChange](#) on page 170

getDeploymentHistoryItems

Retrieves all the deployment history items for a specific environment.

You must specify `projectName` .

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

Example

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

ectool

syntax: ectool getSnapshot <projectName> [optionals]

Example

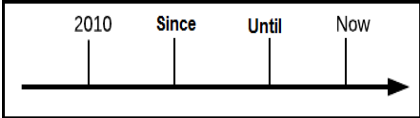
```
ectool getSnapshot "Tutorials" --applicationName "Deploy"
```

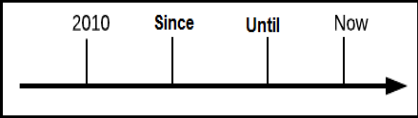
[Back to Top](#)

getEntityChange

Retrieves the entity changes.

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

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

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

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

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

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

Example

If the `entityType` is component:

```
$cmdr->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
<code>entityId</code>	The entity ID. Argument type: UUID
<code>entityType</code>	The entity type. Argument type: String
<code>revisionNumber</code>	The revision number of the entity. Argument type: Integer

Positional arguments

`entityId`, `entityType`, `revisionNumber`

ResponseDetails.

Returns an `entityChange` element.

ec-perl

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

Example

```
$cmdr->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 = <i><Boolean flag -0 1 true false></i> . The default is 0 or false. Argument type: Boolean

Positional arguments

daysToKeep

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->pruneChangeHistory (<daysToKeep>, {<optionals>});

Example

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

ectool

syntax: ectool pruneChangeHistory <daysToKeep> [optionals]

Example

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

[Back to Top](#)

revert

Revert the state of the object to a previous state.

You must enter objectID, objectType, 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
objectId	Object ID. Argument type: UUID

Arguments	Descriptions
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: \$cmdr->revert (<objectID>, <objectType>, <revisionNumber>);

Example

```
$cmdr->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
```

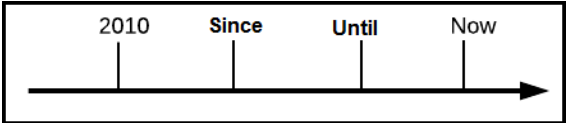
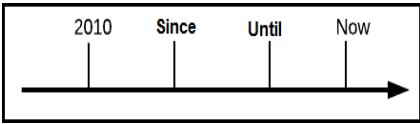
[Back to Top](#)

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
entityType	Type of entity. Argument Type: String

Arguments	Descriptions
<code>modifiedBy</code>	(Optional) The login ID of the user who modified the object. Argument Type: String
<code>timeSince</code>	<p>(Optional) Start of the time interval for changes. ElectricFlow searches for changes since this time. This is the time line:</p>  <p>Argument type: Long</p>
<code>timeUntil</code>	<p>(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 <i>Now</i>. This is the time line:</p>  <p>Argument Type: Long</p>

Positional arguments

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

Response

Returns entity changes during the time interval.

ec-perl

Enter one of these commands:

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

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

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

Example

If the `entityType` is `component`:

```
$cmdr->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](#) on page 172

[copyComponent](#) on page 173

[createComponent](#) on page 175

[deleteComponent](#) on page 177

[getComponent](#) on page 178

[getComponents](#)

[getComponentsinApplicationTier](#) on page 180

[modifyComponent](#)

[removeComponentFromApplicationTier](#) on page 183

addComponentToApplicationTier

Adds the specified component to the specified application tier.

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

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

Arguments	Descriptions
<code>componentName</code>	Name of the component. Argument Type: String
<code>componentProjectName</code>	(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
<code>projectName</code>	The name for the project that must be unique among all projects. Argument Type: String

Arguments	Descriptions
<code>componentName</code>	The name of the component. Argument Type: String
<code>newComponentName</code>	The name of the new component. Argument Type: String
<code>applicationTierName</code>	(Optional) The name of the tier that must be unique within the application. Argument Type: String
<code>description</code>	(Optional) Comment text describing this object. It is not interpreted by ElectricFlow. Argument Type: String
<code>fromApplicationName</code>	(Optional) The name of source application. Argument Type: String
<code>toApplicationName</code>	(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>, {<optional>});
```

Example:

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

ectool

Syntax:

```
ectool copyComponent <projectName> <componentName> <newComponentName> [optional  
s]
```

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
<code>projectName</code>	The name for the project that must be unique among all projects. Argument Type: String
<code>componentName</code>	The name of the component. Argument Type: String
<code>actualParameters</code>	(Optional) The parameters passed as arguments to the application component. Argument Type: Map
<code>applicationName</code>	(Optional) The name of the application where the component is defined. Argument Type: String
<code>credentialName</code>	(Optional) The name of the credential to attach to the component. Argument Type: String
<code>description</code>	(Optional) Comment text describing this object. It is not interpreted by ElectricFlow. Argument Type: String
<code>pluginKey</code>	(Optional) The key of the plugin. Argument Type: String
<code>pluginName</code>	(Optional) The name of the plugin. Argument Type: String
<code>pluginParameters</code>	(Optional) The list of the plugin parameters. Argument Type: Map
<code>reference</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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

Arguments	Descriptions
sourceApplicationName	(Optional) The name of source application. Argument Type: String
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" --sourceComponentName "Backup DB"
--sourceProjectName "Archive" --reference 0
```

[Back to Top](#)

deleteComponent

Deletes a component.

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

Arguments	Descriptions
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>componentName</code>	Name of the component. Argument Type: String
<code>applicationName</code>	(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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>componentName</code>	Name of the component. Argument Type: String
<code>applicationEntityRevisionId</code>	(Optional) The revision ID of the versioned object. Argument type: UUID
<code>applicationName</code>	(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 => "4fa765dd-73f1-11e3-b67e-b0a420524153"});
```

ectool

Syntax:

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

Example:

```
ectool getComponent "Default" "WAR file" --applicationEntityRevisionId "4fa765dd-73f1-11e3-b67e-b0a420524153"
```

[Back to Top](#)

getComponents

Retrieves all components in a project.

You must specify the `projectName`.

Arguments	Descriptions
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>applicationEntityRevisionId</code>	(Optional) The revision ID of the versioned object. Argument type: UUID
<code>applicationName</code>	(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-73f1-11e3-b67e-b0a420524153"});
```

ectool

Syntax:

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

Example:

```
ectool getComponents "Default" --applicationEntityRevisionId "4fa765dd-73f1-11e3-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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application that must be unique among all projects. Argument Type: String
<code>applicationTierName</code>	Name of the tier that must be unique within the application. Argument Type: String
<code>applicationEntityRevisionId</code>	(Optional) The revision ID of the versioned object. Argument type: UUID
<code>includeArtifactDetail</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If set to 1 or <code>true</code> , 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", {applicationEntityRevisionId => "4fa765dd-73f1-11e3-b67e-b0a420524153"});
```

ectool

Syntax:

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

Example:

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

[Back to Top](#)

modifyComponent

Modifies an existing component.

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

Arguments	Descriptions
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>componentName</code>	Name of the component. Argument Type: String
<code>actualParameters</code>	(Optional) Parameters passed as arguments to the application component. Argument Type: Map
<code>applicationName</code>	(Optional) Name of an application to which this component is scoped. Argument Type: String
<code>clearActualParameters</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If this is set to 1 or true, all actual parameters will be removed. Argument type: Boolean

Arguments	Descriptions
<code>credentialName</code>	(Optional) Name of the credential to attach to this component. Argument Type: String
<code>description</code>	(Optional) Comment text describing this component. It is not interpreted by ElectricFlow. Argument Type: String
<code>newName</code>	(Optional) New name of the component. Argument Type: String
<code>pluginKey</code>	(Optional) Key for the plugin. Argument Type: String
<code>pluginName</code>	(Optional) Name of the plugin. Argument Type: String
<code>pluginParameters</code>	(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", newName => "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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application that must be unique among all projects. Argument Type: String
<code>applicationTierName</code>	Name of the tier that must be unique within the application. Argument Type: String
<code>componentName</code>	Name of component. Argument Type: String
<code>componentProjectName</code>	(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 file",  
  {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](#) on page 184[createCredential](#) on page 186[deleteCredential](#) on page 187[detachCredential](#) on page 188[getCredential](#) on page 191[getCredentials](#) on page 192[getFullCredential](#) on page 192[modifyCredential](#) on page 193

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
<code>projectName</code>	The name of the project that must be unique among all projects. Argument type: String
<code>credentialName</code>	Name of the credential in one of these forms: <ul style="list-style-type: none">• relative (for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the request target object.• absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the target object's project. Argument type: String
Locator arguments:	
<code>applicationName</code>	The name of the application process to which the credential is attached. Argument type: String

Arguments	Descriptions
<code>applicationProjectName</code>	The name of the project containing the application. If not specified, it defaults to the Release project name. Argument type: String
<code>componentName</code>	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
<code>pipelineName</code>	The name of the pipeline when attaching a credential to a stage task.
<code>procedureName</code>	The name of a procedure when attaching a credential to a procedure or procedure step. Argument type: String
<code>processName</code>	(Optional) The name of a process when attaching a credential to a process or process step. Argument type: String
<code>processStepName</code>	The name of a process step when attaching a credential to a process step. Argument type: String
<code>releaseName</code>	The name of a Release when attaching a credential to a Release. Argument type: String
<code>scheduleName</code>	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
<code>stageName</code>	The name of the stage when attaching a credential to a task. Argument type: String
<code>stateDefinitionName</code>	The name of the workflow state definition when attaching a credential to a state definition. Argument type: String
<code>stepName</code>	A step name in a procedure or process in the "named" project when attaching a credential to a procedure step. Argument type: String
<code>taskName</code>	The name of the task when attaching a credential to the task. Argument type: String

Arguments	Descriptions
<code>workflowDefinitionName</code>	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
<code>projectName</code>	The name of the project where the credential will be stored. The name must be unique within all projects. Argument type: String
<code>credentialName</code>	The name of the credential. Argument type: String
<code>userName</code>	The user name of the credential. Argument type: String

Arguments	Descriptions
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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String
passwordRecoveryAllowed	(Optional) <i><Boolean flag - 0 1 true false></i> —If 1 or true, the password can be recovered by running <code>getFullCredential</code> 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 => "Use 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
<code>projectName</code>	The name of the project that contains this credential. The project name must be unique among all projects. Argument type: String
<code>credentialName</code>	Name of the credential in one of these forms: <ul style="list-style-type: none"> • relative (for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>")—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: <ul style="list-style-type: none"> • 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
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
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

Arguments	Descriptions
<code>scheduleName</code>	(Optional) The name of the schedule where this credential is attached. Argument type: String
<code>stageName</code>	(Optional) The name of the stage when a credential is attached to a stage task. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the workflow state definition when a credential is attached to a state definition. Argument type: String
<code>stepName</code>	(Optional) A step name in a procedure or process in the "named" project. Argument type: String
<code>taskName</code>	(Optional) The name of the task when a credential is attached to a task. Argument type: String
<code>workflowDefinitionName</code>	(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
<code>projectName</code>	The name of the project that must be unique among all projects. Argument type: String
<code>credentialName</code>	Name of the credential in one of these forms: <ul style="list-style-type: none"> • relative (for example, <code>"cred1"</code>)—The credential is assumed to be in the project that contains the request target object. • absolute (for example, <code>"/projects/BuildProject/credentials/cred1"</code>)—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
<code>projectName</code>	The name of the project that must be unique among all projects. Argument type: String
<code>usableOnly</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If set to <code>1</code> or <code>true</code> , 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	<p>Name of the credential in one of these forms:</p> <ul style="list-style-type: none"> • relative (for example, "cred1")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "/projects/BuildProject/credentials/cred1")—The credential can be from any specified project, regardless of the target object's project. <p>Argument type: String</p>
jobStepId	<p>The unique identifier for the job step that is used to make a project association.</p> <p>Argument type: UUID</p>

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-b0a420524153");

# 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: <ul style="list-style-type: none"> • 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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> 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) <i><Boolean flag - 0 1 true false></i> If this is set to 1 or true, the password can be recovered by running <code>getFullCredential</code> 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](#) on page 195

[setDatabaseConfiguration](#) on page 196

[validateDatabase](#) on page 198

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
<code>customDatabaseDialect</code>	Class name of the Hibernate dialect (<i>advanced use only</i>). The server chooses an appropriate dialect based on the <code>databaseType</code> . Argument Type: String
<code>customDatabaseDriver</code>	Class name of the JDBC driver (<i>advanced use only</i>). The server chooses an appropriate driver based on the <code>databaseType</code> . Argument Type: String
<code>customDatabaseUrl</code>	The JDBC to use (<i>advanced use only</i>). The server composes an appropriate URL. Argument Type: String
<code>databaseName</code>	The name of the database that you want the ElectricFlow server to use. The default is <i>commander</i> . Argument Type: String
<code>databaseType</code>	The type of database that you want the ElectricFlow server to use. Supported database types are: <builtin mysql sqlserver oracle>. The default is <i>builtin</i> . Argument Type: DatabaseType
<code>hostName</code>	The domain name or IP address of the host server machine where the database is running. Argument Type: String

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

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 => "builtin"});
```

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](#) on page 199

[deleteDirectoryProvider](#) on page 202

[getDirectoryProvider](#) on page 203

[getDirectoryProviders](#) on page 204

[modifyDirectoryProvider](#) on page 204

[moveDirectoryProvider](#) on page 208

[testDirectoryProvider](#) on page 209

createDirectoryProvider

Creates a new Active Directory or LDAP directory provider.

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

Arguments	Descriptions
<code>providerName</code>	<p>Name for a LDAP directory provider that must be unique.</p> <p>This human-readable name appears in the user interface to identify users and groups from this provider.</p> <p>Argument type: String</p>
<code>commonGroupNameAttribute</code>	<p>(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 <code>groupNameAttribute</code> or the <code>uniqueGroupNameAttribute</code> is set to <code>distinguishedName</code>, which is not searchable.</p> <p>Argument type: String</p>
<code>description</code>	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
<code>domainName</code>	<p>(Optional) The domain name from which Active Directory servers are automatically discovered.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>emailAttribute</code>	<p>(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.</p> <p>Argument type: String</p>
<code>enableGroups</code>	<p>(Optional) <i><Boolean flag -0 1 true false></i> Determines whether or not to enable external groups for the directory provider. This argument defaults to "true".</p> <p>Argument type: Boolean</p>
<code>fullUserNameAttribute</code>	<p>(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.</p> <p>Argument type: String</p>
<code>groupBase</code>	<p>(Optional) The string is prepended to the <code>basedn</code> to construct the directory domain name (DN) that contains group records.</p> <p>Argument type: String</p>
<code>groupMemberAttributes</code>	<p>(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.</p> <p>Argument type: String</p>
<code>groupMemberFilter</code>	<p>(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 <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> 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.</p> <p>Argument type: String</p>
<code>groupNameAttribute</code>	<p>(Optional) The attribute in a group record that contains the name of the group.</p> <p>Argument type: String</p>
<code>groupSearchFilter</code>	<p>(Optional) This LDAP query is performed in the context of the groups directory to enumerate group records.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>managerDn</code>	<p>(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.</p> <p>Note: This user does not need to be an admin user with modify privileges.</p> <p>Argument type: String</p>
<code>managerPassword</code>	<p>(Optional) If the <code>managerDn</code> property is set, this password is used to authenticate the manager user.</p> <p>Argument type: String</p>
<code>providerType</code>	<p>(Optional) Type string for a directory provider: <ldap activedirectory></p> <p>Argument type: ProviderType</p>
<code>realm</code>	<p>(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").</p> <p>Argument type: String</p>
<code>url</code>	<p>(Optional) The server URL is in the form protocol://host:port/basedn. Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for <code>ldap</code>, 636 for <code>ldaps</code>). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a <code>dc=</code> and separated by commas.</p> <p>Note: Spaces in the <code>basedn</code> must be URL encoded (%20).</p> <p>Argument type: String</p>
<code>useSSL</code>	<p>(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".</p> <p>Note: Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server.</p> <p>Argument type: Boolean</p>

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

Positional arguments

`providerName`

Response

None or status OK message.

ec-perl

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

Example

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

ectool

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

Example

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

[Back to Top](#)

deleteDirectoryProvider

Deletes an Active Directory or LDAP directory provider.

You must specify a `providerName`.

Arguments	Descriptions
providerName	The name of the directory provider that you want to delete. Argument Type: String

Positional arguments

providerName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool deleteDirectoryProvider <providerName>

Example

```
ectool deleteDirectoryProvider 'AD3'
```

[Back to Top](#)

getDirectoryProvider

Retrieves a directory provider by name.

You must specify a providerName.

Arguments	Descriptions
providerName	The name of the directory provider 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: \$cmdr->getDirectoryProvider(<providerName>);

Example

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

ectool

syntax: ectool getDirectoryProvider <providerName>

Example

```
ectool getDirectoryProvider "AD3"
```

[Back to Top](#)

getDirectoryProviders

Retrieves all directory providers.

Arguments	Descriptions
None	

Positional arguments

None

Response

Zero or more [directoryProvider](#) elements.

ec-perl

syntax: \$cmdr->getDirectoryProviders();

Example

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

ectool

syntax: ectool getDirectoryProviders

Example

```
ectool getDirectoryProviders
```

[Back to Top](#)

modifyDirectoryProvider

Modifies an existing LDAP directory provider.

You must specify the `providerName`.

Arguments	Descriptions
<code>providerName</code>	The name of the directory provider that must be unique. Argument Type: String
<code>commonGroupNameAttribute</code>	(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 <code>groupNameAttribute</code> or the <code>uniqueGroupNameAttribute</code> is set to <code>distinguishedName</code> , which is not searchable. Argument Type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> . Argument Type: String
<code>domainName</code>	(Optional) The domain from which Active Directory servers are automatically discovered. Argument Type: String
<code>emailAttribute</code>	(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
<code>enableGroups</code>	(Optional) <i><Boolean flag - 0 1 true false></i> This determines whether or not to enable external groups for the directory provider. Defaults to <code>true</code> . Argument Type: Boolean
<code>fullUserNameAttribute</code>	(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
<code>groupBase</code>	(Optional) This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records. Argument Type: String

Arguments	Descriptions
<code>groupMemberAttributes</code>	<p>(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.</p> <p>Argument Type: String</p>
<code>groupMemberFilter</code>	<p>(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 <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> 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.</p> <p>Argument Type: String</p>
<code>groupNameAttribute</code>	<p>(Optional) The group record attribute that contains the name of the group.</p> <p>Argument Type: String</p>
<code>groupSearchFilter</code>	<p>(Optional) A filter name: this LDAP query is performed in the context of the groups directory to enumerate group records.</p> <p>Argument Type: String</p>
<code>managerDn</code>	<p>(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.</p> <p>Note: This user does not need to be an admin user with modify privileges.</p> <p>Argument Type: String</p>
<code>managerPassword</code>	<p>(Optional) If the <code>managerDn</code> property is set, this password is used to authenticate the manager user.</p> <p>Argument Type: String</p>
<code>newName</code>	<p>(Optional) New name of the directory provider.</p> <p>Argument Type: String</p>
<code>providerType</code>	<p>(Optional) Type string for a directory provider: <code><ldap activedirectory></code>.</p> <p>Argument Type: DirectoryType</p>

Arguments	Descriptions
<code>realm</code>	<p>(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 <code>realm</code> is appended to the user or group name when stored in the ElectricFlow server. For example, <code><user>@dir</code> (where the <code>realm</code> is set to "dir").</p> <p>Argument Type: String</p>
<code>url</code>	<p>(Optional) The LDAP server URL is in the form <code>protocol://host:port/basedn</code>. Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for <code>ldap</code>, 636 for <code>ldaps</code>). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a <code>dc=</code> and separated by commas.</p> <p>Note: Spaces in the <code>basedn</code> must be URL encoded (<code>%20</code>).</p> <p>Argument Type: String</p>
<code>useSSL</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this is set to <code>1</code> or <code>true</code>, SSL is used for communication. The default is <code>1</code> or <code>true</code>.</p> <p>Argument Type: Boolean</p>
<code>userBase</code>	<p>(Optional) This string is prepended to the <code>basedn</code> to construct the directory DN that contains user records.</p> <p>Argument Type: String</p>
<code>userNameAttribute</code>	<p>(Optional) The attribute in a user record that contains the user's account name.</p> <p>Argument Type: String</p>
<code>userSearchFilter</code>	<p>(Optional) This LDAP query is performed in the context of the user directory to search for a user by account name. The string "<code>{0}</code>" is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID</p> <p>Argument Type: String .</p>
<code>userSearchSubtree</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this is set to <code>1</code> or <code>true</code>, ElectricFlow recursively searches the subtree below the user base.</p> <p>Argument Type: Boolean</p>

Arguments	Descriptions
useSSL	<p><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".</p> <p>Note: Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server.</p> <p>Argument Type: Boolean</p>

Positional arguments

providerName

Response

Returns a modified LDAP directory provider element.

ec-perl

syntax: \$cmdr->modifyDirectoryProvider(<providerName>, {<optionals>});

Example

```
$cmdr->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	<p>The name of the directory provider that must be unique.</p> <p>Argument Type: String</p>
beforeProviderName	<p>Moves this directory provider (providerName) to a place before the name specified by this option. If omitted, providerName is moved to the end.</p> <p>Argument Type: String</p>

Positional arguments

providerName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool moveDirectoryProvider <providerName> [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
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 <code>groupNameAttribute</code> or the <code>uniqueGroupNameAttribute</code> is set to <code>distinguishedName</code> , which is not searchable. Argument Type: String
domainName	(Optional) The domain from which Active Directory servers are automatically discovered. Argument Type: String

Arguments	Descriptions
<code>emailAttribute</code>	<p>(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.</p> <p>Argument Type: String</p>
<code>enableGroups</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i> Determines whether or not to enable external groups for the directory provider. Defaults to "true".</p> <p>Argument Type: Boolean</p>
<code>fullUserNameAttribute</code>	<p>(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.</p> <p>Argument Type: String</p>
<code>groupBase</code>	<p>(Optional) This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records.</p> <p>Argument Type: String</p>
<code>groupMemberAttributes</code>	<p>(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.</p> <p>Argument Type: String</p>
<code>groupMemberFilter</code>	<p>(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 <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> 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.</p> <p>Argument Type: String</p>
<code>groupNameAttribute</code>	<p>(Optional) The group record attribute that contains the name of the group.</p> <p>Argument Type: String</p>
<code>groupSearchFilter</code>	<p>(Optional) This LDAP query is performed in the context of the groups directory to enumerate group records.</p> <p>Argument Type: String</p>

Arguments	Descriptions
<code>managerDn</code>	<p>(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.</p> <p>Note: This user does not need to be an admin user with modify privileges.</p> <p>Argument Type: String</p>
<code>managerPassword</code>	<p>(Optional) If the <code>managerDn</code> property is set, this password is used to authenticate the manager user.</p> <p>Argument Type: String</p>
<code>providerType</code>	<p>(Optional) Type string for a directory provider: <code><ldap activedirectory></code>.</p> <p>Argument Type: DirectoryType</p>
<code>realm</code>	<p>(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, <code><user>@dir</code> (where the realm is set to "dir").</p> <p>Argument Type: String</p>
<code>url</code>	<p>(Optional) The LDAP server URL is in the form <code>protocol://host:port/basedn</code>. Protocol is either <code>ldap</code> or <code>ldaps</code> (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for <code>ldap</code>, 636 for <code>ldaps</code>). The <code>basedn</code> is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a <code>dc=</code> and separated by commas.</p> <p>Note: Spaces in the <code>basedn</code> must be URL encoded (<code>%20</code>).</p> <p>Argument Type: String</p>
<code>useDefaults</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to 1 or true, the default values will be used for all fields that are not specified.</p> <p>Argument Type: Boolean</p>
<code>useSSL</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to 1 or true, SSL is used for communication.</p> <p>Argument Type: Boolean</p>

Arguments	Descriptions
<code>userBase</code>	(Optional) This string is prepended to the base DN to construct the directory DN that contains user records. Argument Type: String
<code>userNameAttribute</code>	(Optional) The attribute in a user record that contains the user's account name. Argument Type: String
<code>userSearchFilter</code>	(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
<code>userSearchSubtree</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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: `$cmdr->testDirectoryProvider(<userName>, <password>, {<optionals>});`

Example

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

ectool

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

Example

```
ectool testDirectoryProvider testUser testUserPassword --providerType activeDirectory
--domainName my-company.com
--useDefaults 1
```

```
--managerDn testManager  
--managerPassword testManagerPassword
```

[Back to Top](#)

API Commands - Dynamic Environments

[addResourcePoolToEnvironmentTier](#) on page 214
[addResourceTemplateToEnvironmentTemplateTier](#) on page 215
[addResourceToEnvironmentTemplateTier](#) on page 216
[createEnvironmentTemplate](#) on page 217
[createEnvironmentTemplateTier](#) on page 218
[createEnvironmentTemplateTierMap](#) on page 219
[createHook](#) on page 221
[createResourceTemplate](#) on page 222
[deleteEnvironmentTemplate](#) on page 224
[deleteEnvironmentTemplateTier](#) on page 225
[deleteEnvironmentTemplateTierMapping](#) on page 227
[deleteHook](#) on page 228
[deleteResourceTemplate](#) on page 229
[getAvailableResourcesForEnvironment](#) on page 229
[getEnvironmentTemplate](#) on page 230
[getEnvironmentTemplateTier](#) on page 231
[getEnvironmentTemplateTierMaps](#) on page 232
[getEnvironmentTemplateTiers](#) on page 233
[getEnvironmentTemplates](#) on page 234
[getHook](#) on page 234
[getHooks](#) on page 235
[getProvisionedEnvironments](#) on page 236
[getResourcePoolsInEnvironmentTier](#) on page 237
[getResourceTemplate](#) on page 238
[getResourceTemplates](#) on page 238
[getResourceTemplatesInEnvironmentTemplateTier](#) on page 239
[getResourcesInEnvironmentTemplateTier](#) on page 240
[API Commands - Dynamic Environments](#) on page 213
[modifyEnvironmentTemplate](#) on page 242
[modifyEnvironmentTemplateTier](#) on page 243
[modifyEnvironmentTemplateTierMap](#) on page 244

[modifyHook](#) on page 246

[modifyResourceTemplate](#) on page 247

[provisionEnvironment](#) on page 249

[provisionResourcePool](#) on page 250

[removeResourceFromEnvironmentTemplateTier](#) on page 252

[removeResourcePoolFromEnvironmentTier](#) on page 253

[removeResourceTemplateFromEnvironmentTemplateTier](#) on page 254

[tearDown](#) on page 255

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
<code>resourcePoolName</code>	Name of the resource pool that must be unique among all resource pools. Argument Type: String
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>environmentName</code>	Name of the environment that must be unique among all environments. Argument Type: String
<code>environmentTierName</code>	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

Returns the resource pool and environment tier elements.

ec-perl

```
syntax: $cmdr->addResourcePoolToEnvironmentTier(<resourcePoolName>, <projectName>,  
<environmentName>, <environmentTierName>);
```


Example:

```
$cmdr->addResourcePoolToEnvironmentTier("pool1", "Default", "Production", "Web Server");
```

ectool

syntax:ectool addResourcePoolToEnvironmentTier <resourcePoolName> <projectName>
<environmentName> <environmentTierName>

Example:

```
ectool addResourcePoolToEnvironmentTier "pool1" "Default" "Production" "Web Server"
```

[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
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: \$cmdr->addResourceTemplateToEnvironmentTemplateTier(<projectName>,
<environmentTemplateName>, <resourceTemplateName>, <environmentTemplateTierName>,
{<optionals>});

Example:

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

ectool

syntax: 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: \$cmdr->addResourceToEnvironmentTemplateTier(<resourceName>, <projectName>, <environmentTemplateName>, <environmentTemplateTierName>);

Example:

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

ectool

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

Example:

```
ectool addResourceToEnvironmentTemplateTier "Test station" "Default" "QA" "Tomcat"
```

[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> <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr> . This text is not interpreted by ElectricFlow. Argument type: String

Positional arguments

projectName, environmentTemplateName

Response

Returns an environment template object.

ec-perl

syntax: \$cmdr->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> <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr> . This text is not interpreted by the automation platform. Argument type: String

Arguments	Descriptions
<code>resourceCount</code>	(Optional) The number of resources to be spun from a given resource template. Argument type: Integer
<code>resourceNames</code>	(Optional) List of resources to add to the environment tier. Argument type: Collection
<code>resourceTemplateName</code>	(Optional) Name for the resource template that must be unique among all resource templates. Argument type: String
<code>resourceTemplateProjectName</code>	(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: `$cmdr->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: \$cmdr->createEnvironmentTemplateTierMap(<projectName>, <applicationName>, <environmentProjectName>, <environmentTemplateName>, {<optionals>});

Example:

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

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
<code>hookName</code>	Name of the hook that must be unique among all hooks in the project. Argument Type: String
<code>broadcast</code>	(Optional) Broadcast flag Use this flag to broadcast the hook name in the project. The <code>broadcast</code> value = <i><Boolean flag -0 1 true false></i> . Defaults to <code>true</code> or <code>1</code> . Argument type: Boolean
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> . This text is not interpreted by the automation platform. Argument type: String
<code>hookParameters</code>	(Optional) Parameters that are passed to the procedure. Argument type: Map
<code>hookType</code>	(Optional) Type of the hook: <ul style="list-style-type: none"> • PRE_PROVISIONING • POST_PROVISIONING • PRE_CONFIGUARTION • POST_CONFIGUARTION • PRE_TEARDOWN • POST_TEARDOWN Argument Type: String

Arguments	Descriptions
<code>procedureName</code>	(Optional) Name of the procedure that the hook references. Argument Type: String
<code>procedurePluginKey</code>	(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
<code>procedureProjectName</code>	(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
<code>projectName</code>	(Optional) Project name of the entity that owns the hook. Argument Type: String
<code>resourceTemplateName</code>	(Optional) Name of the resource template. Argument Type: String

Positional arguments

`hookName`

Response

Returns a hook for a resource template.

ec-perl

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

Example:

```
$ec->createHook("config", {hooktype => PRE_CONFIGURATION, procedureName => "Server Start", procedureProjectName => "Servers"});
```

ectool

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

Example:

```
ectool createHook "config" --hookType PRE_CONFIGURATION --procedureName "Server Start" --procedureProjectName "Servers"
```

[Back to Top](#)

createResourceTemplate

Creates a resource template.

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

Arguments	Descriptions
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>resourceTemplateName</code>	Name for the resource template that must be unique among all resource templates. Argument Type: String
<code>cfgMgrParameters</code>	(Optional) Configuration Manager plugin parameters that are passed from the configuration-manager plugin to ElectricFlow. Argument Type: Map
<code>cfgMgrPluginKey</code>	(Optional) Configuration Manager plugin key. Argument Type: String
<code>cfgMgrProcedure</code>	(Optional) Name of the cloud-provider plugin method. Argument Type: String
<code>cfgMgrProjectName</code>	(Optional) Name of the project to which the configuration-manager plugin applies. Argument Type: String
<code>cloudProviderParameters</code>	(Optional) Parameters that are passed from the cloud-provider plugin to ElectricFlow. Argument Type: Map
<code>cloudProviderPluginKey</code>	(Optional) Cloud-provider plugin key. Argument Type: String
<code>cloudProviderProcedure</code>	(Optional) Cloud-provider plugin method name. Argument Type: String
<code>cloudProviderConfig</code>	(Optional) Name of the cloud-provider plugin configuration. Argument Type: String
<code>cloudProviderProjectName</code>	(Optional) Name of the project to which the cloud-provider plugin applies. Argument Type: String

Arguments	Descriptions
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>.</p> <p>This text is not interpreted by the automation platform.</p> <p>Argument type: String</p>

Positional arguments

projectName, resourceTemplateName

Response

Returns a resource template.

ec-perl

syntax: `$cmdr->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	<p>Name of the project that must be unique among all projects.</p> <p>Argument Type: String</p>
environmentTemplateName	<p>Name of the environment template.</p> <p>Argument Type: String</p>

Positional arguments

projectName, environmentTemplateName

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->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: \$cmdr->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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application. Argument Type: String
<code>environmentProjectName</code>	Name for the project to which the environment template belongs. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String

Positional arguments

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

Response

None or a status OK message.

ec-perl

syntax: \$cmdr->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" "Repository"
```

[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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application. Argument Type: String
<code>environmentProjectName</code>	Name for the project to which the environment belongs. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String
<code>applicationTierName</code>	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: \$cmdr->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" "Repository" "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: \$cmdr->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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>resourceTemplateName</code>	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: `$cmdr->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
<code>includePoolUsage</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If this is set to <code>true</code> or <code>1</code> , the pool usage is also retrieved. Argument Type: Boolean
<code>objectTypeToReturn</code>	(Optional) Flag to return resources or resource pools. Valid values are <code>resources</code> or <code>resourcePools</code> . If this is set to <code>true</code> or <code>1</code> , resources or resource pools are also retrieved. Argument Type: String

Response

Returns resource or resource pool objects.

ec-perl

syntax: `$cmdr->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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String

Positional arguments

`projectName`, `environmentTemplateName`

Response

An `environmentTemplate` element.

ec-p erl

syntax: `$cmdr->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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String
<code>environmentTemplateTierName</code>	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: `$cmdr->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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application. Argument Type: String
<code>applicationEntityRevisionId</code>	(Optional) Revision ID of the versioned object. Argument type: UUID
<code>orderByEnvironmentTemplateUsage</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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: `$cmdr->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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String
<code>includeTemplateDetails</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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: `$cmdr->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
<code>projectName</code>	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: `$cmdr->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
<code>hookName</code>	Name of the hook that must be unique among all hooks in the project. Argument Type: String
<code>projectName</code>	(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

Arguments	Descriptions
<code>resourceTemplateName</code>	(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: `$cmdr->getHook(<hookName>, {<optionals>});`

Example:

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

ectool

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

Example:

```
ectool getHook "config" --resourceTemplateName "Servers"
```

[Back to Top](#)

getHooks

Retrieves all the hooks associated with an entity.

Arguments	Descriptions
<code>projectName</code>	(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
<code>resourceTemplateName</code>	(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: \$cmdr->getHooks ({<optionals>});

Example:

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

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-perl

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

Example:

```
$ec->getEnvironmentTemplate ({jobId => "5da765dd-73f1-11e3-b67e-b0a420524153", stageName => "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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>environmentName</code>	Name of the environment that must be unique among all environments. Argument Type: String
<code>environmentTierName</code>	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: `$cmdr->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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>resourceTemplateName</code>	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: `$cmdr->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
<code>projectName</code>	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: `$cmdr->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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String
<code>environmentTemplateTierName</code>	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: \$cmdr->getResourceTemplatesInEnvironmentTemplateTier (<projectName>, <environmentTemplateName>, <environmentTemplateTierName>);

Example:

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

ectool

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

Example:

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

[Back to Top](#)

getResourceTemplatesInEnvironmentTemplateTier

Retrieves all the resources in the specified environment template tier.

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

Arguments	Descriptions
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String
<code>environmentTemplateTierName</code>	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: \$cmdr->getResourcesInEnvironmentTemplateTier (<projectName>, <environmentTemplateName>, <environmentTemplateTierName>);

Example:

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

ectool

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

Example:

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

[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
<code>resourceCount</code>	Number of resources to provision from the specified resource template. Argument Type: Integer
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String
<code>resourceTemplateName</code>	Name of the resource template that must be unique among all resource templates. Argument Type: String
<code>environmentTemplateTierName</code>	Name for the environment template tier that must be unique among all tiers for the environment template. Argument Type: String
<code>resourceTemplateProjectName</code>	(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: \$cmdr->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](#)

modifyEnvironmentTemplate

Modifies an environment template.

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

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

Positional arguments

`projectName`, `environmentTemplateName`

Response

Returns a modified `resourceTemplate` element.

ec-perl

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

Arguments	Descriptions
<code>resourceCount</code>	(Optional) Number of resources to be spun from the resource template. Argument Type: Integer
<code>resourceNames</code>	(Optional) List of resources to add to the environment tier. Argument type: Collection
<code>resourceTemplateName</code>	(Optional) Name for the resource template that must be unique among all resource templates. Argument type: String
<code>resourceTemplateProjectName</code>	(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: `$cmdr->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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application. Argument Type: String
<code>environmentProjectName</code>	Name for the project to which the environment belongs. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String
<code>applicationEntityRevisionId</code>	(Optional) Revision ID of the versioned object. Argument Type: UUID
<code>tierMapName</code>	(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
<code>tierMappings</code>	(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: `$cmdr->modifyEnvironmentTemplateTierMap(<projectName>, <applicationName>, <environmentProjectName>, <environmentTemplateName>, {<optionals>});`

Example:

```
$ec-modifyEnvironmentTemplateTierMap("Default", "Undeploy", "Utilities", "Servers");
```

ectool

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

Example:

```
ectool modifyEnvironmentTemplateTierMap "Default" "Undeploy" "Utilities" "Servers"
```

[Back to Top](#)

modifyHook

Modifies an existing hook in a resource template.

You must specify the `hookName`.

Arguments	Descriptions
<code>hookName</code>	Name of the hook that must be unique among all hooks in the project. Argument Type: String
<code>broadcast</code>	(Optional) Broadcast flag Use this flag to broadcast the hook name in the project. The <code>broadcast</code> value = <i><Boolean flag -0 1 true false></i> . Defaults to <code>true</code> or <code>1</code> . Argument type: Boolean
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> . This text is not interpreted by the automation platform. Argument type: String
<code>hookParameters</code>	(Optional) Parameters that are passed to the procedure. Argument type: Map
<code>hookType</code>	(Optional) Type of the hook: <ul style="list-style-type: none"> • PRE_PROVISIONING • POST_PROVISIONING • PRE_CONFIGUARTION • POST_CONFIGUARTION • PRE_TEARDOWN • POST_TEARDOWN Argument Type: String
<code>newName</code>	New name for the hook.

Arguments	Descriptions
<code>procedureName</code>	Name of the procedure that the hook references. Argument Type: String
<code>procedurePluginKey</code>	(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
<code>procedureProjectName</code>	(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
<code>projectName</code>	(Optional) Project name of the entity that owns the hook. Argument Type: String
<code>resourceTemplateName</code>	(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: `$cmdr->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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>resourceTemplateName</code>	Name for the resource template that must be unique among all resource templates. Argument Type: String
<code>cfgMgrParameters</code>	(Optional) Parameters that are passed from the configuration-manager plugin to ElectricFlow. Argument type: Map
<code>cfgMgrPluginKey</code>	(Optional) Name of the configuration-manager plugin. Argument Type: String
<code>cfgMgrProcedure</code>	(Optional) Name of the configuration-manager plugin method. Argument Type: String
<code>cfgMgrProjectName</code>	(Optional) Name of the project to which the configuration-manager plugin applies. Argument Type: String
<code>cloudProviderParameters</code>	(Optional) Parameters that are passed from the cloud-provider plugin to ElectricFlow. Argument Type: Map
<code>cloudProviderPluginKey</code>	(Optional) Name of the cloud-provider plugin. Parameters that are passed from the cloud-provider plugin to ElectricFlow. Argument Type: String
<code>cloudProviderProcedure</code>	Name of the cloud-provider plugin method. Argument Type: String
<code>cloudProviderProjectName</code>	(Optional) Name of the project to which the cloud-provider plugin applies. Argument Type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> . This text is not interpreted by the automation platform. Argument type: String

Arguments	Descriptions
<code>newName</code>	New name for the resource template. Argument type: String

Positional arguments

`projectName, resourceTemplateName`

Response

None or a status OK message.

ec-perl

syntax: `$cmdr->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
<code>projectName</code>	Name of the project that must be unique among all projects Argument Type: String
<code>environmentName</code>	Name of the environment. Argument Type: String
<code>environmentTemplateName</code>	Name for the environment template that must be unique among all environment templates. Argument Type: String

Arguments	Descriptions
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
keepOnError	(Optional) The keepOnError error flag value is <i><Boolean flag - 0 1 true false></i> . 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: \$cmdr->provisionEnvironment(<projectName>, <environmentName>, <environmentTemplateName>, {<optionals>});

Example:

```
$ec->provisionEnvironment("Default", "Main servers", "Build Server", {keepOnError => 1});
```

ectool

syntax: ectool provisionEnvironment <projectName> <environmentName> <environmentTemplateName> [optionals]

Example:

```
ectool provisionEnvironment "Default" "Main servers" "Build Server" --keepOnError 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
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 <code>keepOnError</code> error flag value is <i><Boolean flag - 0 1 true false></i> . 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: `$cmdr->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
<code>resourceName</code>	Name of the resource that must be unique among all resources. Argument Type: String
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String
<code>environmentTemplateTierName</code>	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: `$cmdr->removeResourceFromEnvironmentTemplateTier(<resourceName>, <projectName>, <environmentTemplateName>, <environmentTemplateTierName>);`

Example:

```
$ec->removeResourceFromEnvironmentTemplateTier("Web Server", "Default", "QA lab", "Tomcat");
```

ectool

syntax: `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
<code>resourcePoolName</code>	Name of the resource pool that must be unique among all resource pools. Argument Type: String
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>environmentName</code>	Name of the environment that must be unique among all environments. Argument Type: String
<code>environmentTierName</code>	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: `$cmdr->removeResourcePoolFromEnvironmentTier (<resourcePoolName>, <projectName>, <environmentName>, <environmentTierName>);`

Example

```
$cmdr->removeResourcePoolFromEnvironmentTier("pool1", "Default", "Production", "Web Server");
```

ectool

syntax: `ectool removeResourcePoolFromEnvironmentTier <resourcePoolName> <projectName> <environmentName> <environmentTierName>`

Example

```
ectool removeResourcePoolFromEnvironmentTier "pool1" "Default" "Production" "Web Server"
```

[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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>environmentTemplateName</code>	Name of the environment template. Argument Type: String
<code>resourceTemplateName</code>	Name of the resource template that must be unique among all resource templates. Argument Type: String
<code>environmentTemplateTierName</code>	Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String
<code>resourceTemplateProjectName</code>	(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: `$cmdr->removeResourceTemplateFromEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <resourceTemplateName>, <environmentTemplateTierName>, {<optionals>});`

Example:

```
$ec->removeResourceTemplateFromEnvironmentTemplateTier("Default", "QA lab", "Tester 1", "App Server", {resourceTemplateProjectName => "QA test"});
```

ectool

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

Example:

```
ectool removeResourceTemplateFromEnvironmentTemplateTier "Default", "QA lab", "Tester 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 <i>seconds</i> to determine how often ectool queries the ElectricFlow server for job status, but this is not an indefinite activity—set the <code>timeout</code> value to extend the <code>pollInterval</code> for longer than 60 seconds if needed. If this option is not specified, <code>runProcedure</code> returns immediately. If it is specified, <code>runProcedure</code> 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 <i>seconds</i> . If <code>pollInterval</code> is specified, this <code>timeout</code> causes <code>runProcedure</code> to stop waiting for the job to complete. It does not stop the job itself. If <code>pollInterval</code> is used and <code>timeout</code> is not used, <code>pollInterval</code> will timeout in 60 seconds. 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: `$cmdr->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](#) on page 256

[deleteEmailConfig](#) on page 258

[getEmailConfig](#) on page 258

[getEmailConfigs](#) on page 259

[modifyEmailConfig](#) on page 260

createEmailConfig

Creates a new email configuration.

You must specify `configName`.

Arguments	Descriptions
<code>configName</code>	The name of your email configuration. Argument type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String

Arguments	Descriptions
<code>mailFrom</code>	(Optional) The email address used as the email sender address for notifications. Argument type: String
<code>mailHost</code>	(Optional) The name of the email server host. Argument type: String
<code>mailPort</code>	(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
<code>mailProtocol</code>	(Optional) This is either SSMTP or SMTP (not case-sensitive). The default is SMTP.
<code>mailUser</code>	(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
<code>mailUserPassword</code>	(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 config object"
```

[Back to Top](#)

deleteEmailConfig

Deletes an email configuration.

You must specify a `configName`.

Arguments	Descriptions
<code>configName</code>	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
<code>configName</code>	The name of your email configuration. Argument Type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> . Argument Type: String
<code>mailFrom</code>	(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: 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
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](#) on page 262

[createEventSubscription](#) on page 266

[deleteEmailNotifier](#) on page 268

[deleteEventSubscription](#) on page 271

[getEmailNotifier](#) on page 273

[getEmailNotifiers](#) on page 276

[getEventSubscription](#) on page 278

[getEventSubscriptions](#) on page 280

[modifyEmailNotifier](#) on page 282

[modifyEventSubscription](#) on page 286

[sendEmail](#) on page 289

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
<code>notifierName</code>	The name of the email notifier. Argument type: String
<code>applicationName</code>	(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
<code>componentName</code>	(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
<code>condition</code>	(Optional) Only send mail if the condition evaluates to "true". The condition is a string subject to property expansion. The notification will <i>not</i> be sent if the expanded string is "false" or "0". If no condition is specified, the notification is <i>always</i> sent. Argument type: String

Arguments	Descriptions
configName	<p>(Optional) If specified, this argument must be the name of an <code>emailConfig</code> object. If it is not specified, the default is the name of the first <code>emailConfig</code> object defined for the ElectricFlow server (<code>emailConfig</code> objects are "ordered" ElectricFlow entities).</p> <p>Note: When using this argument, you must also include the <code>formattingTemplate</code> or the <code>formattingTemplateFile</code> argument .</p> <p>Argument type: String</p>
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
destinations	<p>(Optional) A mandatory argument for a <code>create</code> 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.</p> <p>Argument type: String</p>
environmentNames	<p>(Optional) The environment names.</p> <p>Argument type: Collection</p>
eventType	<p>(Optional) Use one of these values: <code><onStart onCompletion></code>.</p> <ul style="list-style-type: none"> <code>onStart</code>—Triggers an event when the job or job step begins. <code>onCompletion</code>—Triggers an event when the job ends, regardless of the results. <p>The default is <code>onCompletion</code>.</p> <p>Argument type: EventType</p>
flowName	<p>(Optional) The name of the flow.</p> <p>Argument type: String</p>
flowStateName	<p>(Optional) The name of the flow state.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>formattingTemplate</code>	(Optional) A template for formatting email messages when an event [notification] is triggered by the <code>emailNotifier</code> . Make sure the content is formatted correctly, such as no illegal characters or spacing. Argument type: String
<code>formattingTemplateFile</code>	(Optional) This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the <i>formatting template file</i> is read and stored in the "formatting template" field. This is an alternative argument for <code>--formattingTemplate</code> and is useful if the "formatting template" field spans multiple lines.
<code>gateType</code>	(Optional) The type of the gate. Argument type: GateType
<code>groupNames</code>	(Optional) A list of groups that receive the notification. Argument type: Collection
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>notificationType</code>	(Optional) The notification type that is stored to the <code>ec_notificationType</code> property. Argument type: NotificationType
<code>pipelineName</code>	(Optional) The name of the pipeline when a credential attached to a stage task.
<code>procedureName</code>	(Optional) The name of the procedure. When using this argument, you must also enter the <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process that contains of the email notifier. Argument type: String

Arguments	Descriptions
<code>processStepName</code>	(Optional) The name of the process step that contains of the email notifier. Argument type: String
<code>projectName</code>	(Optional) The name of the project. When using this argument, you must also enter the <code>procedureName</code> . Argument type: String
<code>stageName</code>	(Optional) The name of the stage. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) The name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>userNames</code>	(Optional) The names of the users who receive the notification. Argument type: Collection
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workflowName</code>	(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';
```

```

else 'false';}],
    destinations => 'user1@abc.com user2@abc.com emailAlias1@abc.com',
    configName => "testConfiguration",
    projectName => "Project_test",
    procedureName => "Procedure_test",
    formattingTemplate => "Subject: Job started Notification: Job: $[/myJob/jobName] $[
/myEvent/type]
    Job: $[/myJob/jobName] $[/myEvent/type] at $[/myEvent/time]",));

```

ectool

syntax: ectool createEmailNotifier <notifierName> [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
<code>notifierName</code>	The name of the email notifier. Argument type: String
<code>applicationName</code>	(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
<code>componentName</code>	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
<code>flowName</code>	(Optional) The name of the flow. Argument type: String

Arguments	Descriptions
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>gateType</code>	(Optional) The type of the gate. Argument type: GateType
<code>groupNames</code>	A list of groups that receive the notification. Argument type: Collection
<code>jobId</code>	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
<code>jobStepId</code>	The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>procedureName</code>	The name of the procedure. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>processName</code>	The name of the process. Argument type: String
<code>processStepName</code>	The name of the process step. Argument type: String
<code>projectName</code>	The name of the project. When using this argument, you must also enter <code>procedureName</code> . Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	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. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . 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 Fail"});
```

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
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 <code>projectName</code> . Argument type: String
processName	Name of the process. Argument type: String

Arguments	Descriptions
<code>processStepName</code>	Name of the process step. Argument type: String
<code>projectName</code>	Name of the project. Argument type: String
<code>stageName</code>	The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	Name of the state definition. Argument type: String
<code>stateName</code>	Name of the state. Argument type: String
<code>stepName</code>	Name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>taskName</code>	(Optional) The name of the task. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>workflowName</code>	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_test",  
    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
<code>notifierName</code>	The name of the email notifier. Argument type: String
<code>applicationName</code>	(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
<code>componentName</code>	(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
<code>flowName</code>	(Optional) The name of the flow. Argument type: String
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>gateType</code>	(Optional) The type of the gate. Argument type: GateType
<code>groupNames</code>	(Optional) A list of groups that receive the notification. Argument type: Collection

Arguments	Descriptions
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) Name of the process. Argument type: String
<code>processStepName</code>	(Optional) Name of the process step. Argument type: String
<code>projectName</code>	(Optional) The name of the project. When using this argument, you must also enter <code>procedureName</code> . Argument type: String
<code>stageName</code>	(Optional) The name of the stage. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) Name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>taskName</code>	(Optional) The name of the task. Argument type: String

Arguments	Descriptions
<code>userNames</code>	(Optional) A list of names of the users who receives the notification. Argument type: Collection
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workflowName</code>	(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
<code>notifierName</code>	The name of the email notifier. Argument type: String

Arguments	Descriptions
<code>applicationName</code>	(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
<code>componentName</code>	(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
<code>flowName</code>	(Optional) The name of the flow. Argument type: String
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>gateType</code>	(Optional) Type of the gate. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) Name of the process. Argument type: String
<code>processStepName</code>	(Optional) Name of the process step. Argument type: String

Arguments	Descriptions
<code>projectName</code>	(Optional) The name of the project. When using this argument, you must also enter <code>procedureName</code> . Argument type: String
<code>stageName</code>	(Optional) The name of the pipeline stage. Argument type: String
<code>stateDefinitionName</code>	The name of the state definition. Argument type: String
<code>stateName</code>	The name of the state. Argument type: String
<code>stepName</code>	Name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>workflowName</code>	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
<code>applicationName</code>	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
<code>componentName</code>	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
<code>flowName</code>	(Optional) The name of the flow. Argument type: String
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>gateType</code>	(Optional) Type of the gate. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process. Argument type: String

Arguments	Descriptions
<code>processStepName</code>	(Optional) The name of the process step. Argument type: String
<code>projectName</code>	(Optional) The name of the project . When using this argument, you must also enter <code>procedureName</code> . Argument type: String
<code>stageName</code>	(Optional) The name of the pipeline stage. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) The name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>workflowName</code>	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
<code>notifierName</code>	The name of the email notifier. Argument type: String
<code>applicationName</code>	(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
<code>componentName</code>	(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
<code>flowName</code>	(Optional) The name of the flow. Argument type: String
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>gateType</code>	(Optional) Type of the gate. Argument type: String
<code>groupName</code>	The name of the group that receives the notification. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID

Arguments	Descriptions
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>processName</code>	Name of the process. Argument type: String
<code>processStepName</code>	Name of the process step. Argument type: String
<code>projectName</code>	The name of the project . When using this argument, you must also enter <code>procedureName</code> . Argument type: String
<code>stageName</code>	(Optional) The name of the pipeline stage. Argument type: String
<code>stateDefinitionName</code>	The name of the state definition. Argument type: String
<code>stateName</code>	The name of the state. Argument type: String
<code>stepName</code>	Name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>taskName</code>	(Optional) The name of the task. Argument type: String
<code>userName</code>	The names of the users who receives the notification. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>workflowName</code>	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

Arguments	Descriptions
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
procedureName	(Optional) The name of the procedure. When using this argument, you must also enter <code>projectName</code> . 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 <code>procedureName</code> . 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 <code>projectName</code> and <code>procedureName</code> . Argument type: String

Arguments	Descriptions
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workflowName</code>	(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
<code>notifierName</code>	The name of the email notifier. Argument type: String

Arguments	Descriptions
applicationName	<p>(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.</p> <p>Argument type: String</p>
componentName	<p>(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.</p> <p>Argument type: String</p>
condition	<p>(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 <i>a/ways</i> sent.</p> <p>Argument type: String</p>
configName	<p>(Optional) If specified, this argument must be the name of an <code>emailConfig</code> object. If it is not specified, the default is the name of the first <code>emailConfig</code> object defined for the ElectricFlow server (<code>emailConfig</code> objects are "ordered" ElectricFlow entities).</p> <p>Note: When using this argument, you must also include the <code>formattingTemplate</code> or the <code>formattingTemplateFile</code> argument .</p> <p>Argument type: String</p>
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>.</p> <p>Argument type: String</p>
destinations	<p>(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.</p> <p>Note: This argument is mandatory for the "create" operation.</p> <p>Argument type: String</p>
environmentNames	<p>(Optional) The names of the environments.</p> <p>Argument type: Collection</p>

Arguments	Descriptions
eventType	<p>(Optional) Use one of these values: <onStart onCompletion>.</p> <ul style="list-style-type: none"> • <code>onStart</code>—Triggers an event when the job or job step begins. • <code>onCompletion</code>—Triggers an event when the job ends, regardless of the results. <p>The default is <code>onCompletion</code>.</p> <p>Argument type: <code>EventType</code></p>
flowName	<p>(Optional) The name of the flow.</p> <p>Argument type: <code>String</code></p>
flowStateName	<p>(Optional) The name of the flow state.</p> <p>Argument type: <code>String</code></p>
formattingTemplate	<p>(Optional) A template for formatting email messages when an event [notification] is triggered by the <code>emailNotifier</code>. Make sure the content is formatted correctly, such as no illegal characters or spacing.</p> <p>Argument type: <code>String</code></p>
formattingTemplateFile	<p>(Optional) This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the <i>formatting template file</i> is read and stored in the "formatting template" field. This is an alternative argument for <code>formattingTemplate</code> and is useful if the "formatting template" field spans multiple lines.</p>
gateType	<p>(Optional) Type of the gate.</p> <p>Argument type: <code>String</code></p>
groupNames	<p>(Optional) The list of the groups that receives the notification.</p> <p>Argument type: <code>Collection</code></p>
jobId	<p>(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.</p> <p>Argument type: <code>UUID</code></p>
jobStepId	<p>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created.</p> <p>Argument type: <code>UUID</code></p>

Arguments	Descriptions
<code>newName</code>	(Optional) New name of the email notifier. Argument type: String
<code>notificationType</code>	(Optional) The notification type that is stored to the <code>ec_notificationType</code> property. Argument type: NotificationType
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure. When using this argument, you must also enter the <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process that contains of the email notifier. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step that contains of the email notifier. Argument type: String
<code>projectName</code>	(Optional) The name of the project. When using this argument, you must also enter the <code>procedureName</code> . Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) Name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>taskName</code>	(Optional) The name of the task. Argument type: String

Arguments	Descriptions
<code>userNames</code>	(Optional) The names of the users who receives the notification. Argument type: Collection
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workflowName</code>	(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
<code>notifierName</code>	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) 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
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 <code>projectName</code> . Argument type: String
processName	(Optional) The name of the process that contains of the email notifier. Argument type: String

Arguments	Descriptions
<code>processStepName</code>	(Optional) The name of the process step that contains of the email notifier. Argument type: String
<code>projectName</code>	(Optional) The name of the project. When using this argument, you must also enter the <code>procedureName</code> . Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) The name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>taskName</code>	(Optional) The name of the task. Argument type: String
<code>userNames</code>	(Optional) The names of the users who receives the notification. Argument type: Collection
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workflowName</code>	(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.

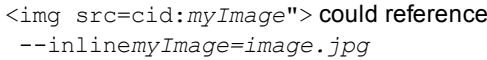
Instead of (or in addition to) specifying a `configName`, any of the configuration options for an email configuration can be specified as options.

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	<p>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.</p> <p>Argument type: Collection</p>
bcc	<p>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.</p> <p>Argument type: Collection</p>
cc	<p>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.</p> <p>Argument type: Collection</p>

Arguments	Descriptions
configName	<p>The name of the email configuration. If no configuration is specified, the configuration named "default" is used.</p> <p>Note: The user must have <i>execute</i> permission on the configuration.</p> <p>Argument type: String</p>
header	<p>An RFC822 email header line (for example: "reply-to: user@host.com"). This option can be specified multiple times.</p> <p>Argument type: Collection</p>
html	<p>The body of a simple HTML message.</p> <p>Argument type: String</p>
htmlFile	<p>Reads the specified client-side file and uses it as the body of a simple HTML message.</p>
inline	<p>Inline attachments in this format: <contentId>=<fileName> [<contentId>=<fileName> ...].</p> <p>One or more inline attachments specified as a <code>contentId</code> and a client-side filename. The filename extension is examined to determine the content-type. The <code>contentId</code> can be referenced in an HTML body using the <code>cid:protocol</code>.</p> <p>For example:  </p> <p>You can enter this argument more than once to add multiple attachments.</p> <p>Argument type: Collection</p>
mailFrom	<p>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.</p> <p>Argument type: String</p>
mailHost	<p>The name of the email server host to use when the <code>configName</code> is not specified. When the email configuration is specified, this value overrides the value in the configuration.</p> <p>Argument type: String</p>
mailPort	<p>The mail server port to use when the <code>configName</code> is not specified. When the email configuration is specified, this value overrides the value in the configuration.</p> <p>Argument type: Integer</p>

Arguments	Descriptions
<code>mailProtocol</code>	<p>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.</p> <p>Argument type: String</p>
<code>mailUser</code>	<p>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.</p> <p>Argument type: String</p>
<code>mailUserPassword</code>	<p>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.</p> <p>Argument type: String</p>
<code>multipartMode</code>	<p>The multipart message mode: <none mixed related mixedRelated></p> <ul style="list-style-type: none"> • <code>none</code>— Non-multipart message • <code>mixed</code>—Single-root multipart element of type "mixed". Texts, inline elements, and attachments will be added to this root element. • <code>related</code>— 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. • <code>mixedRelated</code>—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. <p>Note: <code>multipartMode</code> defaults to <code>none</code> unless there are multiple parts, in which case it defaults to <code>mixedRelated</code>. If both <code>text</code> and <code>html</code> arguments are specified, both values are sent as alternates in a multipart message.</p> <p>Argument type: MultipartMode</p>
<code>raw</code>	<p>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.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>rawFile</code>	Reads the specified client-side file and uses it as the entire mail message, including headers.
<code>subject</code>	The subject of the email message. Argument type: String
<code>text</code>	The body of a simple text message. Argument type: String
<code>textFile</code>	Reads the specified client-side file and uses it as the body of a simple text message.
<code>to</code>	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 </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 </body></html>' \
  --inline image1=image1.jpg image2=image2.jpg \
  --attachment report1.html report2.pdf
```

[Back to Top](#)

API Commands - Environment

[createEnvironment](#) on page 293

[createEnvironmentInventoryItem](#) on page 294

[deleteEnvironment](#) on page 296

[deleteEnvironmentInventoryItem](#) on page 297

[getEnvironment](#) on page 298

[getEnvironmentApplications](#) on page 299

[getEnvironmentInventory](#) on page 299

[getEnvironmentInventoryItem](#) on page 300

[getEnvironmentInventoryItems](#) on page 301

[getEnvironments](#) on page 302

[modifyEnvironment](#) on page 303

[modifyEnvironmentInventoryItem](#) on page 304

[seedEnvironmentInventory](#) on page 305

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 the environment.

Argument Type: Boolean

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"});
```

ectool

Syntax:

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

Example:

```
ectool createEnvironment "Default" "Web Server" --environmentEnabled true --desc  
ription "Web Server"
```

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>,  
<applicationName>, <componentName>, <resourceName>, {<optionals>});
```

Example:

```
$ec->createEnvironmentInventoryItem("Default", "Web Server", "Deploy", "WAR file",  
"local", {artifactName =>"com.ec:myApp", artifactVersion=>"3.1", description =>  
"Required"});
```

ectool

Syntax:

```
ectool createEnvironmentInventoryItem <projectName> <environmentName>  
<applicationName> <componentName> <resourceName> [optionals]
```

Example:

```
ectool createEnvironmentInventoryItem "Default" "Web Server" "Deploy" "WAR file"  
"local"  
-- artifactName "com.ec:myApp" --artifactVersion "3.1" --description "Required"
```

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"
```

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>,  
<applicationName>, <componentName>, <resourceName>, {optionals});
```

Example:

```
$cmdr->deleteEnvironmentInventoryItem("Utilities", "PROD", "Deploy", "WAR file",  
"Server 1", {environmentProjectName => "Default"});
```

ectool

Syntax:

```
ectool deleteEnvironmentInventoryItem <projectName> <environmentName>  
<applicationName> <componentName> <resourceName> [optionals]
```

Example:

```
ectool deleteEnvironmentInventoryItem "Utilities" "PROD" "Deploy" "WAR file" "Se  
rver 1" --environmentProjectName "Default"
```

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"
```

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"
```

getEnvironmentInventory

Retrieves a per-component grouped list of inventory items.

Required Arguments

projectName

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

Argument Type: String

environmentName

Description: Name of the environment.

Argument Type: String

applicationName

Description: Name of the application.

Argument Type: String

Optional Arguments

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>,  
<applicationName>);
```

Example:

```
$ec->getEnvironmentInventory("Default", "Test Server", "Save snapshot");
```

ectool

Syntax:

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

Example:

```
ectool getEnvironmentInventory "Default" "Test Server" "Save snapshot"
```

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

Response

Retrieves an inventory item.

ec-perl

Syntax:

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

Example:

```
$ec->getEnvironmentInventoryItem("Default", "Test Lab", "Save snapshot", "WAR file", "QA server", {c});
```

ectool

Syntax:

```
ectool getEnvironmentInventoryItem <projectName> <environmentName> <applicationName> <componentName> <resourceName> [optionals]
```

Example:

```
ectool getEnvironmentInventoryItem "Default" "Test Lab" "Save snapshot" "WAR file" "QA server" --environmentProjectName "QA only"
```

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

None

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"
```

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"
```

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

Response

Returns an updated environment element.

ec-perl

Syntax:

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

Example:

```
$ec->modifyEnvironment("Default", "Pet Store", {newName => "New Pet Store",  
    description => "Pet store website"});
```

ectool

Syntax:

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

Example:

```
ectool modifyEnvironment "Default" "Pet Store" --newName "New Pet Store"  
--description "Pet store website"
```

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>,
    <applicationName>, <componentName>, <resourceName>, {<optionals>});
```

Example:

```
$ec->modifyEnvironmentInventoryItem("Default", "PROD", "Deploy", "WAR file",
    "Server 1", {artifactVersion => "V3"});
```

ectool

Syntax:

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

Example:

```
ectool modifyEnvironmentInventoryItem "Default" "PROD" "Deploy" "WAR file"
    "Server 1" --artifactVersion "V3"
```

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>,  
  <applicationName>, <applicationProcessName>, <status>, {<optionals>});
```

Example:

```
$ec->seedEnvironmentInventory("Default", "PROD", "Deploy", "Clean up DB",  
  "success", {artifactVersion => "3.0"});
```

ectool

Syntax:

```
ectool seedEnvironmentInventory <projectName> <environmentName>  
  <applicationName> <applicationProcessName> <status> [optionals]
```

Example:

```
ectool seedEnvironmentInventory "Default" "PROD" "Deploy" "Clean up DB"  
  "success" --artifactVersion "3.0"
```

API Commands - Environment Tier

[addResourcesToEnvironmentTier](#) on page 307

[createEnvironmentTier](#) on page 308

[deleteEnvironmentTier](#) on page 309

[getEnvironmentTier](#) on page 310

[getEnvironmentTiers](#) on page 311

[modifyEnvironmentTier](#) on page 312

[removeResourcesFromEnvironmentTier](#) on page 313

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

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

```
$ec->addResourcesToEnvironmentTier("Default", "QA", "Tomcat", {resourceNames => "local"});
```

ectool

Syntax:

```
ectool addResourcesToEnvironmentTier <projectName> <environmentName> <environmentTierName> [optionals]
```

Example:

```
ectool addResourcesToEnvironmentTier "Default" "QA" "Tomcat" --resourceNames "local"
```

createEnvironmentTier

Creates a new environment tier.

Required Arguments

projectName

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

Argument Type: String

environmentName

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

Argument Type: String

environmentTierName

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

Argument Type: String

Optional Arguments

`description`

Description: Comment text describing this object; not interpreted at all by ElectricFlow.

Argument Type: String

`resourceNames`

Description: List of resources to add to the environment tier.

Argument Type: Collection

Response

Returns an environment tier element.

ec-perl

Syntax:

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

Example:

```
$ec->createEnvironmentTier("Default", "PROD", "Web Services", {description => "Required for AWS"});
```

ectool

Syntax:

```
ectool createEnvironmentTier <projectName> <environmentName> <environmentTierName> [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>, <environmentTierName>);
```

Example:

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

ectool

Syntax:

```
ectool deleteEnvironmentTier <projectName> <environmentName> <environmentTierName>
```

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>, <environmentTierName>);
```

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

`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

Response

Retrieves an updated environment tier element.

ec-perl

Syntax:

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

Example:

```
$ec->modifyEnvironmentTier("Default", "QA lab", "Tomcat 1", {newName => "Tomcat 2", description => "New server"});
```

ectool

Syntax:

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

Example:

```
ectool removeResourcesFromEnvironmentTier "Default" "Production" "Apache"
```

API Commands - Gateway and Zone Management

[createGateway](#) on page 314

[deleteGateway](#) on page 316

[getGateway](#) on page 316

[getGateways](#) on page 317

[modifyGateway](#) on page 318

[createZone](#) on page 320

[deleteZone](#) on page 320

[getZone](#) on page 321

[getZones](#) on page 322

[modifyZone](#) on page 322

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
<code>gatewayName</code>	The name of the gateway that must be unique among all gateway names. Argument type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> . Argument type: String
<code>gatewayDisabled</code>	(Optional) <i><Boolean flag - 0 1 true false ></i> If set to 1 or "true", the gateway is disabled. Argument type: Boolean

Arguments	Descriptions
hostName1	(Optional) The domain name or IP of the agent where <i>Resource1</i> resides. This host name is used by <i>Resource2</i> to communicate with <i>Resource1</i> . Do not specify this option is you want to use the host name from the <i>Resource1</i> definition. Argument type: String
hostName2	(Optional) The domain name or IP of the agent where <i>Resource2</i> resides. This host name is used by <i>Resource1</i> to communicate with <i>Resource2</i> . Do not specify this option is you want to use the host name from the <i>Resource2</i> definition. Argument type: String
port1	(Optional) The port number used by <i>Resource1</i> . The default is the port number used by the resource. Argument type: Integer
port2	(Optional) The port number used by <i>Resource2</i> . 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 <code>resourceName2</code> 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 <code>resourceName1</code> 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" --resourceName1 "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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> . 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 <i>Resource1</i> resides. This host name is used by <i>Resource2</i> to communicate with <i>Resource1</i> . Do not specify this option is you want to use the host name from the <i>Resource1</i> definition. Argument type: String
hostName2	(Optional) The domain name or IP address of the agent where <i>Resource2</i> resides. This host name is used by <i>Resource1</i> to communicate with <i>Resource2</i> . Do not specify this option is you want to use the host name from the <i>Resource2</i> definition. Argument type: String
newName	(Optional) New name of the gateway. Argument type: String

Arguments	Descriptions
<code>port1</code>	(Optional) The port number used by <i>Resource1</i> . The default is the port number used by the resource. Argument type: Integer
<code>port2</code>	(Optional) The port number used by <i>Resource2</i> . The default is the port number used by the resource. Argument type: Integer
<code>resourceName1</code>	(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 <code>resourceName2</code> zone. Argument type: String
<code>resourceName2</code>	(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 <code>resourceName1</code> 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
<code>zoneName</code>	The unique name of the zone. Argument type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> . 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 dev group uses."});
```

ectool

syntax: `ectool createZone <zoneName> [optionals]`

Example

```
ectool createZone DevZone --description "Zone containing resources that the dev group 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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> . 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 - Job Management

[abortAllJobs](#) on page 324

[abortJob](#) on page 325

[abortJobStep](#) on page 326

[deleteJob](#) on page 327

[findJobSteps](#) on page 328

[getJobDetails](#) on page 330
[getJobInfo](#) on page 331
[getJobNotes](#) on page 331
[getJobs](#) on page 332
[getJobsForSchedule](#) on page 334
[getJobStatus](#) on page 334
[getJobStepDetails](#) on page 335
[getJobStepStatus](#) on page 336
[getJobSummaries](#) on page 337
[getJobSummary](#) on page 339
[moveJobs](#) on page 340
[runProcedure](#) on page 341
[setJobName](#) on page 345

External Job APIs

[completeJob](#) on page 346
[completeJobStep](#) on page 347
[createJob](#) on page 349
[createJobStep](#) on page 350
[modifyJob](#) on page 360
[modifyJobStep](#) on page 361
[waitForJob](#) on page 368

abortAllJobs

Aborts all running jobs.

Arguments	Descriptions
<code>force</code>	<p>(Optional) <i><Boolean flag - 0 1 true false ></i></p> <p>If this is set to 1 or <code>true</code>, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, executing steps marked "always run".</p> <p>Argument type: Boolean</p>

Arguments	Descriptions
reason	(Optional) A string added to the aborted <code>job/jobstep</code> that describes or records the reason for the abort. The server records this value, but places no meaning on the string, 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
force	(Optional) <i><Boolean flag - 0 1 true false></i> If this is set to 1 or <i>true</i> , the job aborts immediately. A zero value allows jobs to terminate in an orderly way, executing steps marked "always run". Argument type: Boolean

Arguments	Descriptions
reason	(Optional) A string added to the aborted <code>job/jobstep</code> that describes or records the reason for the abort. The server records this value, but places no meaning on the string, 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

Arguments	Descriptions
<code>force</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this is set to <i>1</i> or <i>true</i>, the job aborts immediately. A zero value allows jobs to terminate in an orderly way, for example, executing steps marked "always run".</p> <p>Argument type: Boolean</p>
<code>reason</code>	<p>(Optional) A string added to the aborted <code>job/jobstep</code> that describes or records the reason for the abort. The server records this value, but places no meaning on the string, which is similar to a text description.</p> <p>Argument type: String</p>

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
<code>jobId</code>	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
<code>waitForDeleteToComplete</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If this is set to <i>1</i> or <i>true</i> , ElectricFlow deletes the jobs after completing it. If the value is <i>0</i> or <i>false</i> , 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
<code>jobId</code>	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
<code>jobStepId</code>	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
<code>filters</code>	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
<code>numObjects</code>	<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 <code>numObjects</code> is not specified, all job steps in the list of object IDs are returned. Any and all job steps can be retrieved using the <code>getObjects</code> command. Argument type: Integer
<code>selects</code>	This is an unordered list of property names that specify additional top-level properties to return for each object. See the code example for <code>findObjects</code> 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-b0a420524153"});  
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
<code>jobId</code>	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
<code>structureOnly</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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
<code>jobId</code>	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
<code>jobId</code>	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
<code>firstResult</code>	The first row to return in the results. This is the <i><index number></i> 0-based index that identifies the first element returned from filtered, sorted result set. Argument type: Integer

Arguments	Descriptions
<code>maxResults</code>	This number, <i><result count></i> , sets the maximum number of returned jobs. Argument type: Integer
<code>sortKey</code>	How to sort the results: <i><jobId jobName start finish procedureName></i> . Argument type: String
<code>sortOrder</code>	The order in which to sort the results: <i><ascending descending></i> . Argument type: SortOrder
<code>status</code>	Filter jobs by this status: <i><running completed runnable></i> . 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
<code>projectName</code>	The name for the project that must be unique among all projects. Argument type: String
<code>scheduleName</code>	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
<code>jobId</code>	<p>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.</p> <p>Argument type: UUID</p>

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
<code>jobStepId</code>	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
<code>structureOnly</code>	(Optional) <i><Boolean flag - <0 1 true false></i> cttrue, 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	<p>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.</p> <p>Argument type: UUID</p>

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
<p><code>filters</code></p>	<p>(Optional) Specify filters in a space-separated list: <code>filter1 filter2 ...</code> A list of zero or more filter criteria definitions used to define objects to find.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>Property filter operators are:</p> <pre> 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) </pre> <p>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.</p> <p>Boolean operators are:</p> <pre> not (1 operand) and (2 or more operands) or (2 or more operands) </pre> <p>Argument type: Collection</p>

Arguments	Descriptions
<code>includeLastSuccess</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If this is set to 1 or <code>true</code> , the step will include the last successful job if it was not already included. Argument type: Boolean
<code>maxFailedSteps</code>	(Optional) Maximum number of failed steps to return. Argument type: Integer
<code>maxJobs</code>	(Optional) Maximum number of jobs to return. Argument type: Integer
<code>maxRunningSteps</code>	(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
<code>jobId</code>	(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
<code>jobProperties</code>	(Optional) A comma-separated list of intrinsic job properties to include in the output. Argument type: String
<code>jobStepProperties</code>	(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
<code>sourceProject</code>	Name of the project that contains the jobs you want to move. Argument type: String
<code>destinationProject</code>	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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument type: String
<code>actualParameter</code>	(Optional) Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure. Used in conjunction with <code>procedureName</code> to set the value of the actual parameters. Do not use this argument with <code>scheduleName</code> . Argument type: Map
<code>credentialName</code>	(Optional) <code>credentialName</code> can be one of two forms: relative (for example, <code>"cred1"</code>) - the credential is assumed to be in the project that contains the request target object. absolute (for example, <code>"/projects/BuildProject/credentials/cred1"</code>) - the credential can be from any specified project, regardless of the target object's project.

Arguments	Descriptions
<code>credentials</code>	<p>(Optional) Use the following syntax to specify a credential: <code><credName>=<userName> [<credName>=<userName> ...]</code>.</p> <p>Argument type: Collection</p>
<code>destinationProject</code>	<p>(Optional) This argument is used to specify the name of the destination project.</p> <p>Argument type: String</p>
<code>password</code>	<p>(Optional) The password for the user running the procedure.</p> <p>Argument type: String</p>
<code>pollInterval</code>	<p>(Optional) This argument requires setting a value in <i>seconds</i> to determine how often ectool queries the ElectricFlow server for job status, but this is not an indefinite activity—set the <code>timeout</code> value to extend the <code>pollInterval</code> for longer than 60 seconds if needed.</p> <p>If this option is not specified, <code>runProcedure</code> returns immediately.</p> <p>If it is specified, <code>runProcedure</code> waits until the job completes.</p> <p>Argument type: Integer</p>
<code>priority</code>	<p>(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.</p> <p>Priority values are: <code>low normal high highest</code>.</p> <p>Argument type: JobPriority</p>
<code>procedureName</code>	<p>(Optional) The name for the procedure; must be unique within the project.</p> <p>Argument type: String</p>
<code>scheduleName</code>	<p>(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.</p> <p>Argument type: String</p>

Arguments	Descriptions
timeout	<p>(Optional) This argument requires a value set in <i>seconds</i>.</p> <p>If <code>pollInterval</code> is specified, this <code>timeout</code> causes <code>runProcedure</code> to stop waiting for the job to complete. It does not stop the job itself.</p> <p>If <code>pollInterval</code> is used and <code>timeout</code> is not used, <code>pollInterval</code> will timeout in 60 seconds.</p> <p>Argument type: Integer</p>
userName	<p>(Optional) The name of the user who is running the procedure.</p> <p>Argument type: String</p>

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
<code>jobId</code>	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
<code>newName</code>	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 the command line)
```

```
$cmdr->setJobName("TestJob_252"); (from a step's command)
```

ectool

syntax: `ectool setJobName <jobId> <newName>`

Examples

```
ectool setJobName 4fa765dd-73f1-11e3-b67e-b0a420524153 "Delay Test"_541 (from the command line)
```

```
ectool setJobName "TestJob"_252
```

[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
<code>jobId</code>	<p>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.</p> <p>Argument type: UUID</p>
<code>force</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i> If true, all external steps belonging to the job will be marked "complete".</p> <p>This arguments determines whether all external steps under the job should be recursively marked "complete".</p> <p>Note: If this API is called on a job launched with <code>runProcedure</code>, there is no effect unless <code>force</code> is enabled, in which case only external steps are affected.</p> <p>Argument type: Boolean</p>

Arguments	Descriptions
outcome	<p>(Optional) Overall outcome for a job or step:</p> <ul style="list-style-type: none"> <code>success</code> - The job finished successfully. <code>warning</code> - The job completed with no errors, but encountered some suspicious conditions. <code>error</code> - The job has finished execution with errors. <p>If specified, the outcome overrides any previously propagated outcome value.</p> <p>Argument type: JobOutcome</p>

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
<code>jobStepId</code>	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
<code>exitCode</code>	(Optional) The exit code of a job step. Argument type: Integer
<code>force</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If true, all external steps under the job should be recursively marked "complete". Note: If this API is called on a job launched with <code>runProcedure</code> , there is no effect unless <code>force</code> is enabled, in which case only external steps are affected. Argument type: Boolean
<code>outcome</code>	(Optional) Overall outcome for a job or step: <div> <code>success</code> - The job step finished successfully. <code>warning</code> - The job step completed with no errors, but encountered some suspicious conditions. <code>error</code> - The job step has finished execution with errors. <code>skipped</code> - The job step was skipped. </div> 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
<code>projectName</code>	<p>The name of the project where this job will reside. You must have <i>modify</i> permission on the destination project. <code>projectName</code> or <code>destinationProject</code> must be specified to determine the project where the job is created. If both are specified, <code>destinationProject</code> is preferred.</p> <p>Argument type: String</p>
<code>destinationProject</code>	<p>If specified, determines the project where the job will reside. You must have <i>modify</i> permission on the destination project. <code>projectName</code> or <code>destinationProject</code> must be specified to determine the project where the job is created, <code>destinationProject</code> is preferred.</p> <p>Argument type: String</p>
<code>jobNameTemplate</code>	<p>If specified, the job name will be generated by expanding this argument value.</p> <p>Note: The job name is generated by expanding the <code>jobNameTemplate</code> argument or the <code>jobNameTemplate</code> from the procedure or the system default.</p> <p>Argument type: String</p>
<code>procedureName</code>	<p>If specified, <code>projectName</code> and <code>procedureName</code> are used as a template for the job. You must have <i>execute</i> permission on the procedure.</p> <p>Note: The job name is generated by expanding the <code>jobNameTemplate</code> argument or the <code>jobNameTemplate</code> from the specified procedure or the system default.</p> <p>Argument type: String</p>

Arguments	Descriptions
status	<p>The starting status for the job: <code>pending runnable scheduled running</code>. The <code>status</code> argument determines the "starting" job status. This is useful if you want to immediately go into a specific status without having to use <code>modifyJob</code> to update the status. Defaults to <code>pending</code>.</p> <p>Possible values for <code>status</code>:</p> <ul style="list-style-type: none"> • <code>pending</code>—The job is not yet runnable. • <code>runnable</code>—The job is ready to run. • <code>scheduled</code>—The job is scheduled to run. • <code>running</code>—The job is executing. <p>Argument type: <code>JobStatus</code></p>

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
<code>actualParameter</code>	<p>Specifies the values to pass as parameters to the <i>called</i> procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure.</p> <p>For more information about parameters, click here.</p> <p>Argument type: Map</p>
<code>alwaysRun</code>	<p>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 <code>alwaysRun</code> is a <i><Boolean flag - 0 1 true false></i>. Defaults to "false".</p> <p>Argument type: Boolean</p>
<code>broadcast</code>	<p>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 <code>resourceName</code> argument.</p> <p>The <code>broadcast</code> value = <i><Boolean flag -0 1 true false></i>. This argument is applicable only to command job steps. Defaults to "false".</p> <p>Argument type: Boolean</p>
<code>command</code>	<p>The command to run. This argument is applicable to command job steps only.</p> <p>Argument type: String</p>
<code>comment</code>	<p>The script to execute the functions in this step, which are passed to the step's shell for execution.</p> <p>Argument type: String</p>
<code>condition</code>	<p>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.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>credentialName</code>	<p>The credential to use for impersonation on the agent. <code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p> <p>Argument type: String</p>
<code>credentials</code>	<p>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.</p> <p>Argument type: Collection</p>
<code>errorHandling</code>	<p>Determines what happens to the procedure if the step fails:</p> <ul style="list-style-type: none"> • <code>failProcedure</code> - The current procedure continues, but the overall status is error (default). • <code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete. • <code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure. • <code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run. • <code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps. • <code>ignore</code> - Continues as if the step succeeded. <p>Argument type: ErrorHandling</p>
<code>exclusive</code>	<p><Boolean flag -0 1 true false></p> <p>If set to 1 or true, this job step should acquire and retain this resource exclusively. The default is 0 or false.</p> <p>Note: Setting <code>exclusive</code> sets <code>exclusiveMode</code> to "job".</p> <p>Argument type: Boolean</p>

Arguments	Descriptions
<code>exclusiveMode</code>	<p>Use one of the following options:</p> <ul style="list-style-type: none"> • <code>None</code> - the "default", which does not retain a resource. • <code>Job</code> - 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. • <code>Step</code> - keeps the resource for the duration of the job step. • <code>Call</code> - keeps the resource for the duration of the procedure that called this job step, which is equivalent to 'job' for top level steps. <p>Argument type: <code>ExclusiveMode</code></p>
<code>external</code>	<p>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 <code>createJob</code>). The status of an external job step is set using <code>modifyJobStep</code>, and it can be completed using <code>completeJobStep</code>. The value for external is a <i><Boolean flag - 0 1 true false></i>. Default is "false".</p> <p>Argument type: <code>Boolean</code></p>
<code>jobStepId</code>	<p>The unique ElectricFlow-generated identifier (a 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.</p> <p>If both <code>jobStepId</code> and <code>parentPath</code> are specified, <code>parentPath</code> is preferred.</p> <p>Argument type: <code>UUID</code></p>
<code>jobStepName</code>	<p>The name of the job step.</p> <p>Argument type: <code>String</code></p>
<code>logFileName</code>	<p>A custom log file name produced by running the job step. By default, ElectricFlow assigns a unique name for this file.</p> <p>Argument type: <code>String</code></p>
<code>parallel</code>	<p><i><Boolean flag -0 1 true false></i></p> <p>If set to 1 or <code>true</code>, this job step should run at the same time as adjacent job steps marked to run as parallel. The default is 0 or <code>false</code>.</p> <p>Argument type: <code>Boolean</code></p>

Arguments	Descriptions
<code>parentPath</code>	<p>The path of the parent job step. If both <code>jobStepId</code> and <code>parentPath</code> are specified, <code>parentPath</code> is preferred.</p> <p>Argument type: String</p>
<code>postProcessor</code>	<p>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 <code>postProcessor</code> is a command string for invoking a post-processor program in the platform shell for the resource (<code>cmd</code> for Windows, <code>sh</code> for UNIX).</p> <p>Argument type: String</p>

Arguments	Descriptions
precondition	<p>The <code>precondition</code> property (if it exists) is copied to the job step when the step is created. When the job step is eligible to transition from pending to runnable, the precondition is evaluated. If the precondition result is empty, <code>false</code>, or <code>"0"</code>, the step remains in the pending state. Any other value allows the step to proceed to the runnable state.</p> <p>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.</p> <p>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 <i>precondition</i> is evaluated.</p> <p>A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a <code>"0"</code> or <code>"false"</code> is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p> <p>Argument type: String</p> <p>Precondition example: Assume we defined these 4 steps:</p> <ol style="list-style-type: none"> 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system <p>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.</p> <p>You can achieve optimal step execution order with preconditions:</p> <ul style="list-style-type: none"> • 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 (<code>/myJob/buildInstallerCompleted=1</code>). • Set a precondition in step 4: <code>\$/myJob/buildInstallerCompleted]</code>
procedureName	<p>The name of the procedure for an existing step definition that will be copied to the new job step.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>projectName</code>	<p>The name of the project for an existing step definition that will be copied to the new job step.</p> <p>Argument type: String</p>
<code>releaseExclusive</code>	<p><Boolean flag - 0 1 true false> Declares whether or not this job step will release its resource, which is currently held "exclusively".</p> <p>Note: Setting this flag to "true" is the same as setting <code>releaseMode</code> to <code>release</code>.</p> <p>Argument type: Boolean</p>
<code>releaseMode</code>	<p>Use one of the following options:</p> <ul style="list-style-type: none"> <code>none</code> - the "default" - no action if the resource was not previously marked as "retain." <code>release</code> - 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. <code>releaseToJob</code> - allows a job step to promote a "step exclusive" resource to a Job exclusive resource. <p>Argument type: ReleaseMode</p>
<code>resourceName</code>	<p>Name for the resource; must be unique among all resources.</p> <p>Argument type: String</p>
<code>shell</code>	<p>Where <i>shell</i> 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 (<code>cmd</code> for Windows, <code>sh</code> for UNIX). Applicable to command steps only.</p> <p>Argument type: String</p>

Arguments	Descriptions
status	<p><pending runnable scheduled running> The <code>status</code> argument determines the "starting" job status. This is useful if you want to immediately go into a specific status without having to use <code>modifyJobStep</code> to update the status. Defaults to <code>pending</code>.</p> <p>Possible values for <code>status</code>:</p> <ul style="list-style-type: none"> • <code>pending</code>—The job step is not yet runnable. • <code>runnable</code>—The job step is ready to run. • <code>scheduled</code>—The job step is scheduled to run. • <code>running</code>—The job step is executing. <p>Argument type: <code>JobStatus</code></p>
stepName	<p>The name of an existing step definition that will be copied to the new job step. This defaults to a step in the current procedure if the <code>procedureName</code> and <code>projectName</code> are not specified.</p> <p>Argument type: <code>String</code></p>
subprocedure	<p>The name of the nested procedure to call when this job step runs. If a subprocedure is specified, do not include the <code>command</code> or <code>commandFile</code> options.</p> <p>Argument type: <code>String</code></p>
subproject	<p>If a <code>subprocedure</code> argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.</p> <p>Argument type: <code>String</code></p>
timeLimit	<p>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.</p> <p>Argument type: <code>String</code></p>
timeLimitUnits	<p>Specify <code>hours minutes seconds</code> for time limit units.</p> <p>Argument type: <code>TimeLimitUnits</code></p>

Arguments	Descriptions
<code>workingDirectory</code>	<p>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.</p> <p>Note: If running a job step on a proxy resource, this directory must exist on the proxy target.</p> <p>Argument type: String</p>
<code>workspaceName</code>	<p>The name of the workspace where this job step log files will be stored.</p> <p>Argument type: String</p>

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", external => "1"});
```

```
# Create a job step that calls a subprocedure and passes a parameter credential
```

```
# 'coolProcedure' is a procedure within the Default project with one parameter
```

```
# credential named 'sshCredentialParameter'.
```

```
$cmdr->createJobStep(
  {
    projectName => 'Default',
    subprocedure => 'coolProcedure',
    actualParameter => [
      {
        actualParameterName => 'sshCredentialParameter',
        value => 'sshCredentialParameter'
      }
    ],
  },
)
```



```
credential => [
    {
        credentialName => 'sshCredentialParameter',
        userName => 'sshUser',
        password => 'super_secure_sshPassword'
    }
]
}
);

# Create two parallel job steps and send them to the 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
<code>jobId</code>	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	<p>(Optional) <pending runnable scheduled running></p> <p>This argument determines the current status of the job, and also sets related timing values.</p> <p>Possible values for status:</p> <ul style="list-style-type: none"> • 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. <p>Argument type: JobStatus</p>

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
<code>jobStepId</code>	<p>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.</p> <p>Argument type: UUID</p>
<code>actualParameter</code>	<p>(Optional) Specifies the values to pass as parameters to the <i>called</i> procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure.</p> <p>For more information about parameters, click here.</p> <p>Argument type: Map</p>
<code>alwaysRun</code>	<p>(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 <code>alwaysRun</code> is a <i><Boolean flag - 0 1 true false></i>. Defaults to <code>false</code>.</p> <p>Argument type: Boolean</p>
<code>broadcast</code>	<p>(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 <code>resourceName</code> argument. The <code>broadcast</code> value = <i><Boolean flag -0 1 true false></i>. This argument is applicable only to command job steps. Defaults to <code>false</code>.</p> <p>Argument type: Boolean</p>
<code>command</code>	<p>(Optional) The command to run. This argument is applicable to command job steps only.</p> <p>Argument type: String</p>
<code>comment</code>	<p>(Optional) The script to execute the functions in this step, which are passed to the step's shell for execution.</p> <p>Argument type: String</p>
<code>condition</code>	<p>(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.</p> <p>Argument type: String</p>

Arguments	Descriptions
errorHandling	<p>(Optional) Determines what happens to the procedure if the step fails:</p> <ul style="list-style-type: none"> • <code>failProcedure</code> - The current procedure continues, but the overall status is error (default). • <code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete. • <code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure. • <code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run. • <code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps. • <code>ignore</code> - Continues as if the step succeeded. <p>Argument type: ErrorHandling</p>
exclusive	<p>(Optional) If set to 1, indicates this job step should acquire and retain this resource exclusively. The value for <code>exclusive</code> is a <i><Boolean flag -0 1 true false></i>. Defaults to false. Note: Setting <code>exclusive</code>, sets <code>exclusiveMode</code> to "job".</p> <p>Argument type: Boolean</p>
exclusiveMode	<p>(Optional) Use one of the following options:</p> <ul style="list-style-type: none"> • <code>None</code> - the "default", which does not retain a resource. • <code>Job</code> - 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. • <code>Step</code> - keeps the resource for the duration of the job step. • <code>Call</code> - keeps the resource for the duration of the procedure that called this job step, which is equivalent to 'job' for top level steps. <p>Argument type: ExclusiveMode</p>
logFileName	<p>(Optional) A custom log file name produced by running the job step. By default, ElectricFlow assigns a unique name for this file.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>parallel</code>	<p>(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 <code>parallel</code> is a <i><Boolean flag -0 1 true false></i>. Defaults to false.</p> <p>Argument type: Boolean</p>
<code>postProcessor</code>	<p>(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 <code>postProcessor</code> is a command string for invoking a post-processor program in the platform shell for the resource (<code>cmd</code> for Windows, <code>sh</code> for UNIX).</p> <p>Argument type: String</p>

Arguments	Descriptions
precondition	<p>(Optional) The <code>precondition</code> property (if it exists) is copied to the job step when the step is created. When the job step is eligible to transition from pending to runnable, the precondition is evaluated. If the precondition result is empty, <code>false</code>, or <code>0</code>, the step remains in the pending state. Any other value allows the step to proceed to the runnable state.</p> <p>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.</p> <p>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 <i>precondition</i> is evaluated.</p> <p>A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a <code>"0"</code> or <code>"false"</code> is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p> <p>Argument type: String</p> <p>Precondition example: Assume we defined these 4 steps:</p> <ol style="list-style-type: none"> 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system <p>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.</p> <p>You can achieve optimal step execution order with preconditions:</p> <ul style="list-style-type: none"> • 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 (<code>/myJob/buildInstallerCompleted=1</code>). • Set a precondition in step 4: <code>\$/myJob/buildInstallerCompleted]</code>
releaseExclusive	<p>(Optional) <Boolean flag - 0 1 true false> Declares whether or not this job step will release its resource, which is currently held "exclusively".</p> <p>Note: Setting this flag to <code>true</code> is the same as setting <code>releaseMode</code> to <code>release</code>.</p> <p>Argument type: Boolean</p>

Arguments	Descriptions
<code>releaseMode</code>	<p>(Optional) Use one of the following options:</p> <ul style="list-style-type: none"> • <code>none</code> - the "default" - no action if the resource was not previously marked as "retain." • <code>release</code> - 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. • <code>releaseToJob</code> - allows a job step to promote a "step exclusive" resource to a Job exclusive resource. <p>Argument type: ReleaseMode</p>
<code>resourceName</code>	<p>(Optional) Name for the resource; must be unique among all resources.</p> <p>Argument type: String</p>
<code>shell</code>	<p>(Optional) Where <i>shell</i> 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 (<code>cmd</code> for Windows, <code>sh</code> for UNIX). Applicable to command steps only.</p> <p>Argument type: String</p>
<code>status</code>	<p>(Optional) <code><pending runnable scheduled running></code> The <code>status</code> argument determines the "starting" job status. This is useful if you want to immediately go into a specific status without having to use <code>modifyJobStep</code> to update the status. Defaults to <code>pending</code>.</p> <p>Possible values for <code>status</code>:</p> <ul style="list-style-type: none"> • <code>pending</code>—The job step is not yet runnable. • <code>runnable</code>—The job step is ready to run. • <code>scheduled</code>—The job step is scheduled to run. • <code>running</code>—The job step is executing. <p>Argument type: JobStatus</p>
<code>subprocedure</code>	<p>(Optional) The name of the nested procedure to call when this job step runs. If a subprocedure is specified, do not include the <code>command</code> or <code>commandFile</code> options.</p> <p>Argument type: String</p>

Arguments	Descriptions
subproject	(Optional) If a <code>subprocedure</code> 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 <code>hours minutes seconds</code> 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
<code>jobId</code>	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
<code>timeout</code>	(Optional) The number of seconds to wait before giving up on a request. The default is 60 seconds. Argument type: Integer
<code>finalStatus</code>	(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

To wait until a job has a status of 'completed' with no timeout:

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

To wait until a job has a *completed* status with a timeout of 30 seconds:

```
$cmdr->waitForJob("4fa765dd-73f1-11e3-b67e-b0a420524153", 30);
```

To wait for a job that has a *running* status with no timeout:

```
$cmdr->waitForJob("4fa765dd-73f1-11e3-b67e-b0a420524153", undef, "running");
```

[Back to Top](#)

API Commands - Parameter Management

[attachParameter](#) on page 369
[createActualParameter](#) on page 371
[createFormalParameter](#) on page 374
[deleteActualParameter](#) on page 379
[deleteFormalParameter](#) on page 380
[detachParameter](#) on page 382
[getActualParameter](#) on page 384
[getActualParameters](#) on page 387
[getFormalParameter](#) on page 390
[getFormalParameterOptions](#) on page 392
[getFormalParameters](#) on page 393
[modifyActualParameter](#) on page 395
[modifyFormalParameter](#) on page 397
[validateFormalParameters](#) on page 400

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
<code>projectName</code>	The name of the project that must be unique among all projects. Argument Type: String
<code>procedureName</code>	The name of the procedure to which the parameter is attached. Argument Type: String
<code>stepName</code>	The name of the procedure step to which the parameter is attached. Argument Type: String

Arguments	Descriptions
<code>formalParameterName</code>	The name of the parameter. Argument Type: String
<code>applicationName</code>	(Optional) The name of the application to which the parameter is attached. Argument Type: String
<code>componentApplicationName</code>	(Optional) The name of the component in the application to which the parameter is attached. Argument Type: String
<code>componentName</code>	(Optional) The name of the component to which the parameter is attached. Argument Type: String
<code>gateType</code>	(Optional) The type of the gate to which the parameter is attached: POST or PRE. Argument Type: GateType
<code>pipelineName</code>	(Optional) The name of the pipeline to which the parameter is attached. Argument Type: String
<code>processName</code>	(Optional) The name of the process to which the parameter is attached. Argument Type: String
<code>processStepName</code>	(Optional) The name of the process step to which the parameter is attached. Argument Type: String
<code>stageName</code>	(Optional) The name of the task to which the parameter is attached. Argument Type: String
<code>stateDefinitionName</code>	(Optional) The name of the workflow state definition to which the parameter is attached. Argument Type: String
<code>taskName</code>	(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" --stageName "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
<code>projectName</code>	The name of the project containing the procedure. The project name must be unique among all projects. Argument type: String
<code>procedureName</code>	The name of the procedure containing the step. Argument type: String
<code>stepName</code>	The name of the step that calls a subprocedure. Argument type: String
<code>actualParameterName</code>	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
<code>applicationName</code>	(Optional) The name of the application when the actual parameter is on an application process step. Argument type: String
<code>componentName</code>	(Optional) The name of the component when the actual parameter is on a component process step. Argument type: String
<code>flowName</code>	(Optional) The name of the flow. Argument type: String
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>processName</code>	(Optional) The name of the process when the actual parameter is on a process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when if the actual parameter is on a process step. Argument type: String
<code>releaseName</code>	(Optional) The name of the Release if the actual parameter is on a Release. Argument type: String
<code>scheduleName</code>	(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 directory",  
    {value => "local"});
```

ectool

syntax: ectool <projectName> <procedureName> <stepName> <actualParameterName>
[optionals]

Example

```
ectool createActualParameter "Default" "Take Snapshot" "Retrieve" "Source directory"  
    --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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> 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
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) <i><Boolean flag - 0 1 true false></i> If 1 or true, this parameter is required. The procedure will not execute unless a value is given for it. Argument type: Boolean
stateDefinitionName	(Optional) The name of the state definition. Argument type: String
stateName	(Optional) The name of a workflow state. Argument type: String
type	(Optional) The <i>type</i> can be any string value. This argument is used primarily by the web interface to represent custom form elements. When the type is the <i>credential</i> 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, `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
<code>projectName</code>	The name of the project that contains this actual parameter. The name must be unique among all projects. Argument type: String
<code>procedureName</code>	The name of the procedure that contains the step with this parameter. Argument type: String
<code>stepName</code>	The name of the step that contains the actual parameter. Argument type: String
<code>actualParameterName</code>	The name of the actual parameter to delete. Argument type: String
<code>applicationName</code>	(Optional) The name of the application when the actual parameter is on an application process step. Argument type: String
<code>componentName</code>	(Optional) The name of the component when the actual parameter is on a component process step. Argument type: String
<code>processName</code>	(Optional) The name of the process when the actual parameter is on a process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the actual parameter is on a process step. Argument type: String
<code>releaseName</code>	(Optional) The name of the Release if the actual parameter is used in a Release. Argument type: String

Arguments	Descriptions
<code>scheduleName</code>	(Optional) The name of the schedule containing the actual parameter. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>transitionDefinitionName</code>	(Optional) The name of the transition definition. Argument type: String
<code>workflowDefinitionName</code>	(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 Version');
```

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
<code>projectName</code>	The name of the project that contains the procedure or parameter to delete. The name must be unique among all projects. Argument type: String
<code>procedureName</code>	The name of the procedure that contains this parameter. Argument type: String
<code>formalParameterName</code>	The name of the formal parameter to delete. Argument type: String
<code>applicationName</code>	(Optional) The name of the application when the formal parameter is on an application process step. Argument type: String
<code>componentName</code>	(Optional) The name of the component when the formal parameter is on a component process step. Argument type: String
<code>flowName</code>	(Optional) The name of the flow. Argument type: String
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>processName</code>	(Optional) The name of the process when the formal parameter is on a process. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the formal parameter is on a process step. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the workflow state. Argument type: String
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String

Arguments	Descriptions
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 summary"});
```

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
<code>projectName</code>	The name of the project that contains this parameter. The name must be unique among all projects. Argument type: String
<code>procedureName</code>	The name of the procedure that contains this parameter. Argument type: String
<code>stepName</code>	The name of the step where this parameter is currently attached. Argument type: String
<code>formalParameterName</code>	The name of the formal parameter to detach. Argument type: String
<code>applicationName</code>	(Optional) The name of the application. Argument type: String
<code>componentApplicationName</code>	(Optional) The name of the component in the application. Argument type: String
<code>componentName</code>	(Optional) The name of the component. Argument type: String
<code>gateType</code>	(Optional) The type of the gate. Argument type: GateType
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>processName</code>	The name of the process that contains this parameter. Argument type: String
<code>processStepName</code>	The name of the step in the process.. Argument type: String
<code>stageName</code>	(Optional) The name of the stage. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String

Arguments	Descriptions
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

Arguments	Descriptions
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
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

Arguments	Descriptions
<code>procedureName</code>	<p>(Optional) The name of the procedure when the actual parameter is on a procedure step.</p> <p>If you need actual parameters on a step, the <code>projectName</code>, <code>procedureName</code>, and <code>stepName</code> arguments must be used together to specify the step.</p> <p>Argument type: String</p>
<code>processName</code>	<p>(Optional) The name of the process when the actual parameter is on a process step.</p> <p>Argument type: String</p>
<code>processStepName</code>	<p>(Optional) The name of the process step when the actual parameter is on a process step.</p> <p>Argument type: String</p>
<code>projectName</code>	<p>(Optional) The name of the project to query for the parameter on a schedule or procedure step.</p> <p>If you need actual parameters on a step, the <code>projectName</code>, <code>procedureName</code>, and <code>stepName</code> arguments must be used together to specify the step.</p> <p>Argument type: String</p>
<code>releaseName</code>	<p>(Optional) The name of the Release if the actual parameter is on it.</p> <p>Argument type: String</p>
<code>scheduleName</code>	<p>(Optional) The name of the schedule to query for the parameter on a schedule.</p> <p>Argument type: String</p>
<code>stateDefinitionName</code>	<p>(Optional) The name of the workflow state definition.</p> <p>Argument type: String</p>
<code>stateName</code>	<p>(Optional) The name of the workflow state.</p> <p>Argument type: String</p>
<code>stepName</code>	<p>(Optional) The name of the step to query for the parameter on the step.</p> <p>Argument type: String</p>
<code>transitionDefinitionName</code>	<p>(Optional) The name of the workflow transition definition.</p> <p>Argument type: String</p>

Arguments	Descriptions
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
<code>applicationEntityRevisionId</code>	(Optional) The revision ID of the versioned object. Argument type: UUID
<code>applicationName</code>	(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
<code>componentName</code>	(Optional) The name of the component when the actual parameters are on a component process step. Argument type: String
<code>flowName</code>	(Optional) Name of the flow that must be unique within the project. Argument Type: String
<code>flowRuntimeName</code>	(Optional) Name of the flow runtime. Argument Type: String
<code>flowRuntimeStateId</code>	(Optional) The ID of the flow runtime state. Argument Type: UUID
<code>flowRuntimeStateName</code>	(Optional) Name of the flow state. Argument Type: String
<code>flowStateName</code>	(Optional) Name of the flow state that must be unique within the flow. Argument Type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>procedureName</code>	(Optional) The name of the procedure containing the parameters. To find parameters on a step, you must use <code>projectName</code> , <code>procedureName</code> , and <code>stepName</code> to specify the step. Argument type: String

Arguments	Descriptions
<code>processName</code>	(Optional) The name of the process when the actual parameters are on a process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the actual parameters are on a process step. Argument type: String
<code>projectName</code>	(Optional) The name of the project containing the parameters. To find parameters on a step, you must use <code>projectName</code> , <code>procedureName</code> , and <code>stepName</code> to specify the step. Argument type: String
<code>releaseName</code>	(Optional) The name of the Release if the actual parameter is on it. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule containing parameters. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the workflow state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the workflow state. Argument type: String
<code>stepName</code>	(Optional) The name of the step containing parameters. To find parameters on a step, you must use <code>projectName</code> , <code>procedureName</code> , and <code>stepName</code> to specify the step. Argument type: String
<code>transitionDefinitionName</code>	(Optional) The name of the workflow transition definition. Argument type: String
<code>transitionName</code>	(Optional) The name of the workflow transition. Argument type: String
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workflowName</code>	(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
<code>projectName</code>	The name of the project containing the procedure. The name must be unique among all projects. Argument type: String
<code>procedureName</code>	The name of the procedure containing the formal parameter. In ElectricFlow 5.0 and later, this argument is optional. Argument type: String
<code>formalParameterName</code>	The name for the formal parameter that is used when the procedure is invoked to specify a value for the parameter. Argument type: String
<code>applicationEntityRevisionId</code>	(Optional) The revision ID of the versioned object. Argument type: UUID

Arguments	Descriptions
<code>applicationName</code>	(Optional) The name of the application when the formal parameter is on an application process step. Argument type: String
<code>componentName</code>	(Optional) The name of the component when the formal parameter is on a component process step. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline when the formal parameter is on a pipeline. Argument type: String
<code>processName</code>	(Optional) The name of the process when the formal parameter is on a process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the formal parameter is on a process step. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the workflow state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the workflow state. Argument type: String
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workflowName</code>	(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
<code>formalParameterName</code>	The name of the parameter for which the options are requested. Argument type: String
<code>procedureName</code>	The name of the procedure containing the formal parameter. Argument type: String
<code>actualParameters</code>	(Optional) Parameters passed as arguments to the script. Argument type: String
<code>configurationParameters</code>	(Optional) Configuration parameter values passed as arguments to the script. Argument type: Map

Arguments	Descriptions
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 Project"
--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
<code>projectName</code>	The name of the project containing the parameters. The name must be unique among all projects. Argument type: String
<code>applicationEntityRevisionId</code>	(Optional) The revision ID of the versioned object. Argument type: UUID
<code>applicationName</code>	(Optional) The name of the application when the formal parameters are on an application process step. Argument type: String
<code>componentName</code>	(Optional) The name of the component when the formal parameters are on a component process step. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline when the formal parameter is on a pipeline. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure when the formal parameters are on a procedure. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process when the formal parameters are on a process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the formal parameter is on a process step. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the workflow state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the workflow state. Argument type: String

Arguments	Descriptions
stepName	(Optional) The name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . 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
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

Arguments	Descriptions
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 level",
    {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
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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String
expansionDeferred	(Optional) <i><Boolean flag - 0 1 true false></i> If this is set to 1 or <code>true</code> , 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 <code>false</code> , 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
label	(Optional) The display label. Argument type: String

Arguments	Descriptions
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
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) <i><Boolean flag - 0 1 true false></i> 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
stateDefinitionName	(Optional) The name of the state definition. Argument type: String
stateName	(Optional) The name of a workflow state. Argument type: String
type	(Optional) The <i>type</i> can be any string value. This argument is used primarily by the web interface to represent custom form elements. When the type is the <i>credential</i> 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
<code>procedureName</code>	The name of a procedure. Argument type: String
<code>actualParameters</code>	(Optional) Parameters passed as arguments to the script.. Argument type: String
<code>configurationParameters</code>	(Optional) Configuration parameter values passed as arguments to the script. Argument type: Map
<code>credentials</code>	(Optional) Credentials to be used in the script. Argument type: Collection
<code>pluginName</code>	(Optional) The name of a plugin. Argument type: String
<code>projectName</code>	(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

[abortPipelineRun](#) on page 403

[completeManualTask](#) on page 404

[completeRelease](#) on page 405

[createDeployer](#) on page 406
[createDeployerApplication](#) on page 406
[createDeployerConfiguration](#) on page 408
[createNote](#) on page 410
[createPipeline](#) on page 411
[createRelease](#) on page 412
[createStage](#) on page 413
[createTask](#) on page 414
[deleteNote](#) on page 418
[deletePipeline](#) on page 419
[deletePipelineRun](#) on page 419
[deleteRelease](#) on page 420
[deleteStage](#) on page 421
[deleteTask](#) on page 422
[getDeployerApplication](#) on page 422
[getDeployerConfiguration](#) on page 424
[getDeployerConfigurations](#) on page 425
[getNote](#) on page 426
[getNotes](#) on page 427
[getPipeline](#) on page 428
[getPipelineRuntimeDetails](#) on page 429
[getPipelineRuntimes](#) on page 429
[getPipelineStageRuntimeDeployerTasks](#) on page 431
[getPipelineStageRuntimeTasks](#) on page 432
[getPipelines](#) on page 433
[getRelease](#) on page 433
[getReleaseInventory](#) on page 434
[getReleases](#) on page 435
[getStage](#) on page 437
[getStages](#) on page 438
[getTask](#) on page 439
[getTasks](#) on page 440
[getWaitingTasks](#) on page 441
[modifyDeployer](#) on page 442
[modifyDeployerApplication](#) on page 443
[modifyDeployerConfiguration](#) on page 444

[modifyNote](#) on page 446

[modifyPipeline](#) on page 447

[modifyRelease](#) on page 448

[modifyStage](#) on page 450

[modifyTask](#) on page 451

[removeDeployerApplication](#) on page 455

[removeDeployerConfiguration](#) on page 456

[runPipeline](#) on page 457

[startRelease](#) on page 458

[validateDeployer](#) on page 458

[waitForFlowRuntime](#) on page 460

abortPipelineRun

Aborts a pipeline run.

Arguments	Descriptions
<code>flowRuntimeId</code>	(Optional) The ID of the flow runtime. Argument Type: UUID
<code>flowRuntimeName</code>	(Optional) The name of the flow runtime. Argument Type: String
<code>force</code>	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , the pipeline aborts immediately. A zero value allows the running task in the pipeline stage to terminate. Argument Type: Boolean
<code>projectName</code>	(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 => "Production"});
```

ectool

syntax: `ectool abortPipelineRun [optionals]`

Example

```
ectool abortPipelineRun --flowRuntimeName "Final Release" --projectName "Production"
```

[Back to Top](#)

completeManualTask

Completes the manual task.

You must specify the `flowRuntimeId`, `stageName`, and `taskName` arguments.

Arguments	Descriptions
<code>flowRuntimeId</code>	ID of the flow runtime. Argument Type: UUID
<code>stageName</code>	Name of the stage. Argument Type: String
<code>taskName</code>	Name of the task. Argument Type: String
<code>action</code>	(Optional) The type of action to be taken on the pipeline stage manual task. Valid values are <code>completed</code> and <code>failed</code> . Argument type: <code>ManualProcessStepStatus</code>
<code>actualParameters</code>	(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called process. Argument Type: Map
<code>evidence</code>	(Optional) User-supplied evidence provided while performing an action on the pipeline stage manual task or stage gate task. Argument type: String
<code>gateType</code>	(Optional) The type of the gate. Argument type: <code>GateType</code>

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", "Confirm version");
```

ectool

syntax: ectool completeManualTask <flowRuntimeId> <stageName> <taskName> [optionals]

Example

```
ectool completeManualTask 4fa765dd-73f1-11e3-b67e-b0a420524153 "Final Deploy" "Confirm 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
<code>projectName</code>	The name of the project that must be unique among all projects. Argument Type: String
<code>deployerName</code>	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
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) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , 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 <code>false</code> or <code>0</code> , the pipeline deploys all the application processes, components, and artifacts in the application, referred to as a <i>full run</i> . Argument Type: Boolean
snapshotName	(Optional) The name of the application snapshot. Argument Type: String
stageArtifacts	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , 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 <code>false</code> or <code>0</code> , 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
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 <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called process. Argument Type: Map

Arguments	Descriptions
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
releaseName	(Optional) The name of the Release. Argument Type: String
skipDeploy	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , 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](#)

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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>pipelineName</code>	Name of the pipeline that must be unique among all projects. Argument Type: String
<code>description</code>	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
<code>enabled</code>	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , the pipeline is enabled. Argument type: Boolean
<code>type</code>	(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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>releaseName</code>	Name of the Release. Argument Type: String
<code>actualParameters</code>	(Optional) Actual parameters. Argument Type: Map
<code>description</code>	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
<code>pipelineName</code>	(Optional) Name of the pipeline. Argument Type: String
<code>pipelineProjectName</code>	(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
<code>plannedEndDate</code>	(Optional) The date when this release is expected to end (for example, 2016-05-15). Argument Type: String
<code>plannedStartDate</code>	(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", pipelineName => "Daily Update"});
```

ectool

syntax: `ectool createRelease <projectName> <releaseName> [optionals]`

Example

```
ectool createRelease "Default" "Production" --description "Pet Shop Web Site" --pipelineName "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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>stageName</code>	Name of the stage. Argument Type: String
<code>afterStage</code>	(Optional) The stage that is placed after the new stage. Argument Type: String
<code>beforeStage</code>	(Optional) The stage that is placed before the new stage. Argument Type: String
<code>description</code>	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
<code>pipelineName</code>	(Optional) Name of the pipeline to which the stage belongs. Argument type: String

Positional arguments

`projectName, stageName`

Response

Returns a stage object.

ec-perl

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

Example

```
$ec->createStage("Default", "Preflight", {afterStage => "Automated Tests"});
```

ectool

syntax: `ectool createStage <projectName> <stageName> [optionals]`

Example

```
ectool createStage "Default" "Preflight" --afterStage "Automated Tests"
```

[Back to Top](#)

createTask

Creates a new task for a task container.

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

Arguments	Descriptions
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>taskName</code>	Name of the task. Argument Type: String
<code>actualParameters</code>	(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called process. Argument Type: Map
<code>advancedMode</code>	(Optional) <i><Boolean flag - 0 1 true false></i> When <code>advancedMode</code> is set to <code>true</code> or <code>1</code> , advanced mode is enabled so that all validations for the <code>applicationName</code> , <code>environmentName</code> , <code>processName</code> , or <code>snapshotName</code> will be skipped. These objects can be represented using the <code>"\$[]"</code> notation and later expanded at the pipeline runtime. For example, when <code>advancedMode</code> is enabled, the system does not validate that the environment exists or whether it is mapped to an application. Argument type: Boolean
<code>afterTask</code>	(Optional) The task that is placed after the named task. Argument Type: String

Arguments	Descriptions
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
description	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
enabled	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , the task is enabled. Argument type: Boolean
environmentName	(Optional) Name of the environment to create from a template. When the <code>environmentTemplateName</code> is specified, the <code>environmentName</code> 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 <code>environmentTemplateName</code> is specified, the <code>environmentName</code> can be specified using the "\$[]" notation. It is expanded at runtime. Argument Type: String
environmentTemplateName	(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: ErrorHandling
gateType	(Optional) The type of the gate. Argument Type: GateType

Arguments	Descriptions
instruction	(Optional) Instruction associated with the task. Argument Type: String
keepOnError	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , the system is set to keep environment if an error occurs. The default is <code>false</code> or <code>0</code> . 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
skippable	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , the task can be skipped in the pipeline. Argument type: Boolean
snapshotName	(Optional) Name of the snapshot associated with the application. When <code>advancedMode</code> is set to <code>true</code> or <code>1</code> , advanced mode is enabled so that all validations for the <code>applicationName</code> , <code>environmentName</code> , <code>processName</code> , or <code>snapshotName</code> will be skipped. These objects can be represented using the "\$[]" notation and later expanded at the pipeline runtime. For example, when <code>advancedMode</code> 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 <code>hh:mm</code> 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

Arguments	Descriptions
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
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

```
$ec->createTask("Default", "Save results", {beforeTask => "Save log file"});
```

ectool

syntax:ectool createTask <projectName> <taskName> [optionals]

Example

```
ectool createTask "Default" "Save results" --beforeTask "Save log file"
```

[Back to Top](#)

deleteNote

Deletes a note associated with an entity.

You must specify `note`.

Arguments	Descriptions
<code>note</code>	Notes provided about the entity. Argument Type: String
<code>projectName</code>	(Optional) Project name of the entity that owns the note. Argument Type: String
<code>releaseName</code>	(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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>pipelineName</code>	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
<code>flowRuntimeId</code>	(Optional) The ID of the flow runtime. Argument Type: UUID
<code>flowRuntimeName</code>	(Optional) The name of the flow runtime. Argument Type: String

Arguments	Descriptions
<code>projectName</code>	(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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>releaseName</code>	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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>stageName</code>	Name of the stage. Argument Type: String
<code>pipelineName</code>	(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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>taskName</code>	Name of the task. Argument Type: String
<code>gateType</code>	(Optional) The type of the gate. Argument Type: GateType
<code>pipelineName</code>	(Optional) Name of the pipeline to which the stage belongs. Argument type: String
<code>stageName</code>	(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](#)

getDeployerApplication

Retrieves the application used in the Release by name.

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

Arguments	Descriptions
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application. Argument Type: String
<code>applicationProjectName</code>	(Optional) The name of the project containing specified application. If this argument is not specified, the default is the Release project. Argument Type: String
<code>releaseName</code>	(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

Arguments	Descriptions
<code>applicationProjectName</code>	(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
<code>releaseName</code>	(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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String

Arguments	Descriptions
<code>applicationName</code>	(Optional) Name of the application. Argument Type: String
<code>applicationProjectName</code>	(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
<code>releaseName</code>	(Optional) The name of the Release. Argument Type: String
<code>stageName</code>	(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 configurationn objects.

ec-perl

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

Example

```
$ec->getDeployerConfigurations("Default", {applicationProjectName => "Quarterly Summaries"});
```

ectool

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

Example

```
ectool getDeployerConfigurations "Default" --applicationProjectName "Quarterly Summaries"
```

[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
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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>pipelineName</code>	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
firstResult	(Optional) First row of the results to return, based on how the results are paginated. Argument Type: Integer

Arguments	Descriptions
<code>flowRuntimeId</code>	(Optional) ID of the flow runtime. Argument Type: UUID
<code>maxResults</code>	(Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer
<code>pipelineName</code>	(Optional) Name of the pipeline. Argument Type: String
<code>projectName</code>	(Optional) Name of the project that must be unique among all projects. Argument Type: String
<code>releaseId</code>	(Optional) ID of the release. Argument Type: String
<code>sortKey</code>	(Optional) How to sort the results. Argument Type: String
<code>sortOrder</code>	(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 => "Default"});
```

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
<code>flowRuntimeId</code>	ID of the flow runtime. Argument Type: UUID
<code>stageName</code>	The name of the stage. Argument Type: String
<code>taskName</code>	The name of the Deployer task. Argument Type: String
<code>firstResult</code>	(Optional) First row of the results to return, based on how the results are paginated. Argument Type: Integer
<code>maxResults</code>	(Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer
<code>sortKey</code>	(Optional) How to sort the results. Argument Type: String
<code>sortOrder</code>	(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", "Deploy WAR file", {firstResult => 2, maxResults => 200});
```

ectool

syntax:ectool getPipelineStageRuntimeTasks <flowRuntimeId> <stageName> <taskName> [optionals]

Example

```
ectool getPipelineStageRuntimeTasks 4fa765dd-73f1-11e3-b67e-b0a420524165 "PROD" "Deploy 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
<code>flowRuntimeId</code>	ID of the flow runtime. Argument Type: UUID
<code>firstResult</code>	(Optional) First row of the results to return, based on how the results are paginated. Argument Type: Integer
<code>maxResults</code>	(Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer
<code>sortKey</code>	(Optional) How to sort the results. Argument Type: String
<code>sortOrder</code>	(Optional) The order in which the results are sorted. Argument Type: SortOrder
<code>stageName</code>	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, {firstResult => 2, maxResults => 200});
```

ectool

syntax:ectool getPipelineStageRuntimeTasks <flowRuntimeId> [optionals]

Example

```
ectool getPipelineStageRuntimeTasks 4fa765dd-73f1-11e3-b67e-b0a420524165 --firstResult 2 --maxResults 200
```

[Back to Top](#)

getPipelines

Retrieves all the pipelines.

You must specify the `projectName`.

Arguments	Descriptions
<code>projectName</code>	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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>releaseName</code>	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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>releaseName</code>	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
<code>projectName</code>	(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	<p>(Optional) A list of zero or more filter criteria definitions used to define objects to find.</p> <p>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.</p> <p>Two types of filters:</p> <p>"property filters" are used to select objects based on the value of the object's intrinsic or custom property.</p> <p>"boolean filters" ("and", "or", "not") are used to combine one or more filters using boolean logic.</p> <p>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.</p> <p>Property filter operators are:</p> <pre> 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) </pre> <p>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.</p> <p>Boolean operators are:</p> <pre> not (1 operand) and (2 or more operands) </pre>

Arguments	Descriptions
	<p>or (2 or more operands)</p> <p>Argument type: Collection</p>
<code>firstResult</code>	<p>(Optional) First row of the results to return, based on how the results are paginated.</p> <p>Argument Type: Integer</p>
<code>maxResults</code>	<p>(Optional) The number of rows to return, based on how the results are paginated.</p> <p>Argument Type: Integer</p>
<code>sortKey</code>	<p>(Optional) How to sort the results.</p> <p>Argument Type: String</p>
<code>sortOrder</code>	<p>(Optional) The order in which the results are sorted.</p> <p>Argument Type: SortOrder</p>

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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>taskName</code>	Name of the task. Argument Type: String
<code>gateType</code>	(Optional) Type of the gate. Argument Type: GateType
<code>pipelineName</code>	(Optional) Name of the pipeline to which the stage belongs. Argument type: String
<code>stageName</code>	(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
<code>flowRuntimeId</code>	The ID of the flow runtime. Argument type: UUID
<code>stageName</code>	Name of the stage. Argument type: String
<code>deployerTaskName</code>	(Optional) The name of the deployer task. Argument type: String
<code>firstResult</code>	(Optional) In the paginated results, the first row to return. Argument type: Integer
<code>maxResults</code>	(Optional) In the paginated results, the number of rows to return. Argument type: Integer
<code>sortKey</code>	(Optional) How to sort the results: <jobId jobName start finish procedureName>. Argument type: String
<code>sortOrder</code>	(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 => start, sortOrder => ascending});
```

ectool

syntax: ectool getWaitingTasks <projectName> <stageName> [optionals]

Example

```
ectool getWaitingTasks "5da765dd-73f1-11e3-b67e-b0a420524153" "PROD" --sortKey start --sortOrder ascending
```

[Back to Top](#)

modifyDeployer

Modifies an existing Deployer.

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

Arguments	Descriptions
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>deployerName</code>	Name of the Deployer that must be unique within the project. Argument Type: String
<code>newName</code>	(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 Cart"});
```

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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application. Argument Type: String
<code>applicationProjectName</code>	(Optional) The name of the project containing specified application. If not specified, it is defaulted to the Release project name. Argument Type: String
<code>orderIndex</code>	(Optional) The application deployment order, starting at 1. Argument type: Integer
<code>processName</code>	(Optional) Name of the process. Argument type: String
<code>releaseName</code>	(Optional) Name of the Release. Argument type: String
<code>smartDeploy</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If this is set to 1 or true, the <code>smart deploy</code> option is used when the application is deployed. Argument type: Boolean
<code>snapshotName</code>	(Optional) Name of the snapshot. Argument type: String

Arguments	Descriptions
stageArtifacts	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>When this argument is set to <code>true</code> or <code>1</code>, artifact staging is enabled for the application process. ElectricFlow retrieves artifacts that will be deployed in an application run before the deployment starts.</p> <p>When this argument is set to <code>false</code> or <code>0</code>, artifact staging is disabled.</p> <p>Argument Type: Boolean</p>

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	<p>Name of the project that must be unique among all projects.</p> <p>Argument Type: String</p>
applicationName	<p>Name of the application.</p> <p>Argument Type: String</p>

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 <code>actualParameterName</code> and a value. The <code>actualParameterName</code> 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) <i><Boolean flag - 0 1 true false></i> 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
releaseName	(Optional) The name of the Release. Argument Type: String
skipDeploy	(Optional) <i><Boolean flag - 0 1 true false></i> 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](#)

modifyNote

Modifies a note associated with an entity.

You must specify the `noteName` argument.

Arguments	Descriptions
noteName	Name of the note. Argument Type: String
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
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) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , 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

```
$ec->modifyPipeline("Default", "Web Site Deploy", {newName => "Web Site Update"});
```

ectool

syntax: ectool modifyPipeline <projectName> <pipelineName> [optionals]

Example

```
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 <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called process. Argument Type: Map
clearActualParameters	(Optional) <i><Boolean flag - 0 1 true false></i> If set to <code>true</code> , tasks remove all actual parameters. Argument type: Boolean

Arguments	Descriptions
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>.</p> <p>This text is not interpreted by the automation platform.</p> <p>Argument type: String</p>
newName	<p>(Optional) New name for an existing pipeline.</p> <p>Argument Type: String</p>
pipelineName	<p>(Optional) Name of the pipeline.</p> <p>Argument Type: String</p>
pipelineProjectName	<p>(Optional) Name of the project containing specified pipeline. If this argument is not specified, the default is the Release project name.</p> <p>Argument Type: String</p>
plannedEndDate	<p>(Optional) The date when this release is expected to end (for example, 2016-05-15).</p> <p>Argument Type: String</p>
plannedStartDate	<p>(Optional) The date when this release is expected to begin (for example, 2016-05-15)</p> <p>Argument Type: String</p>

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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>stageName</code>	Name of the stage. Argument Type: String
<code>afterStage</code>	(Optional) The stage that is placed after the new stage. Argument Type: String
<code>beforeStage</code>	(Optional) The stage that is placed before the new stage. Argument Type: String
<code>description</code>	(Optional) Comment text describing this object; it is not interpreted at all by ElectricFlow. Argument Type: String
<code>newName</code>	(Optional) New name for the existing stage. Argument Type: String
<code>pipelineName</code>	(Optional) Name of the pipeline to which the stage belongs. Argument type: String

Positional arguments

`projectName, stageName`

Response

Returns a modified stage object.

ec-perl

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

Example

```
$ec->modifyStage("Default", "PROD", {pipelineName => "Daily build"});
```

ectool

syntax: `ectool modifyStage <projectName> <stageName> [optionals]`

Example

```
ectool modifyStage "Default" "PROD" --pipelineName "Daily build"
```

[Back to Top](#)

modifyTask

Modifies an existing task.

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

Arguments	Descriptions
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>taskName</code>	Name of the task. Argument Type: String
<code>actualParameters</code>	(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called process. Argument Type: Map
<code>advancedMode</code>	(Optional) <i><Boolean flag - 0 1 true false></i> When <code>advancedMode</code> is set to <code>true</code> or <code>1</code> , advanced mode is enabled so that all validations for the <code>applicationName</code> , <code>environmentName</code> , <code>processName</code> , or <code>snapshotName</code> will be skipped. These objects can be represented using the "\$[]" notation and later expanded at the pipeline runtime. For example, when <code>advancedMode</code> is enabled, the system does not validate that the environment exists or whether it is mapped to an application. Argument type: Boolean
<code>afterTask</code>	(Optional) The task that is placed after the new task. Argument Type: String
<code>approvers</code>	(Optional) A list of task approvers who receive the notification. Argument Type: Collection
<code>beforeTask</code>	(Optional) The task that is placed before the new task. Argument Type: String
<code>clearActualParameters</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If set to <code>true</code> or <code>1</code> , all the actual parameters are removed from the flow. Argument type: Boolean

Arguments	Descriptions
description	(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String
enabled	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , the task is enabled. Argument type: Boolean
environmentName	(Optional) Name of the environment to create from a template. When the <code>environmentTemplateName</code> is specified, the <code>environmentName</code> 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 <code>environmentTemplateName</code> is specified, the <code>environmentName</code> can be specified using the "\$[]" notation. It is expanded at runtime. Argument Type: String
environmentTemplateName	(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: ErrorHandling
gateType	(Optional) The type of the gate. Argument Type: GateType
instruction	(Optional) Instruction associated with the task. Argument type: String
keepOnError	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , the system is set to keep environment if an error occurs. The default is <code>false</code> or <code>0</code> . Argument type: Boolean

Arguments	Descriptions
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
skippable	(Optional) <i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , the task can be skipped in the pipeline. Argument type: Boolean
snapshotName	(Optional) Name of the snapshot associated with the application. When <code>advancedMode</code> is set to <code>true</code> or <code>1</code> , advanced mode is enabled so that all validations for the <code>applicationName</code> , <code>environmentName</code> , <code>processName</code> , or <code>snapshotName</code> will be skipped. These objects can be represented using the "\$[]" notation and later expanded at the pipeline runtime. For example, when <code>advancedMode</code> 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 <code>hh:mm</code> 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

Arguments	Descriptions
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

```
$ec->modifyTask("Default", "Save results", {newName => "Save output"});
```

ectool

syntax: ectool modifyTask <projectName> <taskName> [optionals]

Example

```
ectool modifyTask "Default" "Save results" --newName "Save output"
```


[Back to Top](#)

removeDeployerApplication

Removes a Deployer application for a Release.

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

Arguments	Descriptions
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application. Argument Type: String
<code>applicationProjectName</code>	(Optional) Name of the project containing specified application. If it is not specified, the default is the Release project name. Argument Type: String
<code>releaseName</code>	(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", {applicationProjectName => "Software tools"});
```

ectool

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

Example

```
ectool removeDeployerApplication "Default" "Verify versions" --applicationProjectName "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
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application. Argument Type: String
<code>stageName</code>	Name of the stage of a pipeline attached to a release if specified. Argument Type: String
<code>applicationProjectName</code>	(Optional) Name of the project containing the application. If this argument is not specified, the default is the Release project name. Argument Type: String
<code>releaseName</code>	(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", {applicationProjectName => "Online store"});
```

ectool

syntax: `ectool removeDeployerConfiguration <projectName> <applicationName> <stageName> [optionals]`

Example

```
ectool removeDeployerConfiguration "Default" "Shopping Cart" "QA" --applicationProjectName "Online store"
```

[Back to Top](#)

runPipeline

Runs the specified pipeline.

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

Arguments	Descriptions
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>pipelineName</code>	Name of the pipeline that must be unique among all projects. Argument Type: String
<code>actualParameters</code>	(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called process. Argument Type: Map
<code>credentials</code>	(Optional) The credentials to be used in the task. Argument type: Collection
<code>priority</code>	(Optional) Priority of jobs launched by the workflow. Argument Type: JobPriority
<code>startingState</code>	(Optional) Name of the starting state. 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", {startingState => "Green"});
```

ectool

syntax: `ectool runPipeline <projectName> <pipelineName> [optionals]`

Example

```
ectool runPipeline "Default" "Web Site Update" --startingState "Green"
```

[Back to Top](#)

startRelease

Starts a Release.

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

Arguments	Descriptions
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>releaseName</code>	Name of the Release. Argument Type: String
<code>priority</code>	(Optional) The priority of jobs launched by the release.. Argument Type: JobPriority
<code>startingStage</code>	(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" --startingState "Checkout"
```

[Back to Top](#)

validateDeployer

Validates the Deployer configuration.

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

Arguments	Descriptions
<code>projectName</code>	Name of the project that must be unique among all projects. Argument Type: String
<code>applicationName</code>	Name of the application. Argument Type: String
<code>stageName</code>	Name of the stage of a pipeline attached to a release if specified. Argument Type: String
<code>actualParameters</code>	(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called process. Argument Type: Map
<code>applicationProjectName</code>	(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
<code>environmentName</code>	(Optional) Name of the environment to create from a template. Argument Type: String
<code>environmentProjectName</code>	(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
<code>environmentTemplateName</code>	(Optional) Name of the environment template to use. Argument Type: String
<code>environmentTemplateName</code>	(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
<code>releaseName</code>	(Optional) Name of the Release. Argument Type: String
<code>validationType</code>	(Optional) The type of validation to perform (PARAM for parameter, ENV for environment, or 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 Site"});
```

ectool

syntax: ectool validateDeployer <projectName> <applicationName> <stageName> [optionals]

Example

```
ectool validateDeployer "Default" "Make WAR file" "Checkin" --releaseName "Web Site"
```

[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
<code>flowRuntimeId</code>	The ID of the flow runtime for the pipeline. Argument type: UUID
<code>timeout</code>	(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

[createPlugin](#) on page 461

[deletePlugin](#) on page 462

[getPlugin](#) on page 463

[getPlugins](#) on page 464

[installPlugin](#) on page 464

[modifyPlugin](#) on page 466

[promotePlugin](#) on page 467

[uninstallPlugin](#) on page 468

createPlugin

Creates a plugin from an existing project.

You must specify a `key`, `version`, and `projectName`.

Arguments	Descriptions
<code>key</code>	Version independent name for the plugin. Argument type: String
<code>version</code>	Version of the plugin. Argument type: String
<code>projectName</code>	Name of the project. Argument type: String
<code>author</code>	(Optional) Name of the plugin author. Argument type: String

Arguments	Descriptions
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
label	<p>(Optional) Label that appears in plugin lists.</p> <p>Argument type: String</p>

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	<p>The name of the plugin.</p> <p>Argument type: String</p>

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
<code>pluginName</code>	<p>The name of the plugin to find. If the name is specified without a version number, the currently promoted version is returned if possible.</p> <p>Argument type: String</p>

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
<code>url</code>	<p>The location of the plugin JAR file to install.</p> <p>If the location refers to a file on the client machine, the file will be uploaded to the server.</p> <p>If the location refers to a remote accessible file (for example, through <code>http://url</code>), the server will download it.</p> <p>If the location is a <code>file:</code> reference, the file will be read directly from the specified location on the server file system.</p> <p>Argument type: String</p>
<code>disableProjectTracking</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to <code>1</code> or <code>true</code> 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.</p> <p>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.</p> <p>Argument type: Boolean</p>
<code>force</code>	<p>(Option) <i><Boolean flag - 0 1 true false></i></p> <p>Specifying <code>0</code> or <code>false</code> causes an existing plugin with the same key and version to be overwritten with the new plugin contents, otherwise an error is returned.</p> <p>Argument type: Boolean</p>
<code>reducedDetailChangeHistory</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>Use this argument for large projects containing over 20,000 audited objects with Change Tracking enabled.</p> <p>When this argument is set to <code>true</code> or <code>1</code>, 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.</p> <p>Set this argument to <code>false</code> or <code>0</code> 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.</p> <p>Argument type: Boolean</p>

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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String

Arguments	Descriptions
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

Arguments	Descriptions
<code>promoted</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>The new value of the promoted flag for the specified plugin.</p> <p>The default is <code>true</code>, which means the plugin will be promoted.</p> <p>If you want to demote the plugin, use the value of "0" or <code>false</code>.</p> <p>Argument type: Boolean</p>

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
<code>pluginName</code>	<p>The name of the plugin to uninstall. If the name is specified without a version number, the currently promoted version is used if possible.</p> <p>Argument type: String</p>
<code>timeout</code>	<p>(Optional) The maximum amount of time to spend waiting for this operation to complete.</p> <p>Argument type: Long</p>

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](#) on page 469

[createStep](#) on page 471

[deleteProcedure](#) on page 478

[deleteStep](#) on page 479

[getProcedure](#) on page 480

[getProcedures](#) on page 481

[getStep](#) on page 481

[getSteps](#) on page 482

[modifyProcedure](#) on page 483

[modifyStep](#) on page 485

[moveStep](#) on page 491

createProcedure

Creates a new procedure for an existing project.

You must specify `projectName` and `procedureName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String
<code>procedureName</code>	The name of the procedure that must be unique within the project. Argument type: String
<code>credentialName</code>	(Optional) The name of the credential specified in one of these formats: <ul style="list-style-type: none"> • relative(for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the requested target object. • absolute(for example, "<i>projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the project for the target object. Argument type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String
<code>jobNameTemplate</code>	(Optional) Template used to determine the default name of jobs launched from a procedure. Argument type: String
<code>resourceName</code>	(Optional) The name of the default resource or pool used in steps run by this procedure. Argument type: String
<code>timeLimit</code>	(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 <code>runProcedure</code> (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
<code>timeLimitUnits</code>	(Optional) Time limit units are <code>hours</code> <code>minutes</code> <code>seconds</code> . Argument type: TimeLimitUnits

Arguments	Descriptions
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: \$cmdr->createProcedure(<projectName>, <procedureName>, {<optionals>});

Example

```
$cmdr->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

Arguments	Descriptions
<code>procedureName</code>	The name for a procedure that must be unique within the project. Argument type: String
<code>stepName</code>	Name of the step that must be unique within the procedure. Argument type: String
<code>actualParameters</code>	(Optional) Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure. For more information about parameters, click here . Argument type: Map
<code>afterProcedureStep</code>	(Optional) If specified, the procedure step will be placed after the named procedure step. Argument type: String
<code>alwaysRun</code>	(Optional) <i><Boolean flag -0 1 true false></i> . The default is <code>false</code> . If this value is set to <code>1</code> or <code>true</code> , 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
<code>beforeProcedureStep</code>	(Optional) If specified, the procedure step will be placed before the named procedure step. Argument type: String
<code>broadcast</code>	(Optional) <i><Boolean flag -0 1 true false></i> . The default is <code>false</code> . If this value is set to <code>1</code> or <code>true</code> , 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 <code>resourceName</code> argument. This argument is applicable only to command steps. Argument type: Boolean
<code>command</code>	(Optional) The command to run. This argument is applicable only to command steps. Argument type: String
<code>comment</code>	(Optional) The script to execute the functions of this step, which are passed to the step's shell for execution. Argument type: String

Arguments	Descriptions
<code>commandFile</code>	<p>(Optional) This option is supported only in Perl and ectool bindings - it is not a part of the XML protocol.</p> <p>Contents of the <i>command file</i> is read and stored in the "command" field. This is an alternative argument for <code>command</code> and is useful if the "command" field spans multiple lines. The <code>commandFile</code> value is the actual <i>command file</i> text. This argument is applicable to command steps only.</p>
<code>condition</code>	<p>(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.</p> <p>Argument type: String</p>
<code>credentialName</code>	<p>(Optional) The credential to use for impersonation on the agent.</p> <p>Name of the credential in one of these forms:</p> <ul style="list-style-type: none"> • relative (for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the target object's project. <p>Argument type: String</p>
<code>description</code>	<p>(Optional) A plain text or HTML description for this object.</p> <p>If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>

Arguments	Descriptions
errorHandling	<p>(Optional) Determines what happens to the procedure if the step fails:</p> <ul style="list-style-type: none"> 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. <p>Argument type: ErrorHandling</p>
exclusive	<p>(Optional) <i><Boolean flag -0 1 true false></i>.</p> <p>If set to 1 or true, this step should acquire and retain this resource exclusively. The default is false.</p> <p>Note: When you set exclusive, exclusiveMode is set to "job".</p> <p>Argument type: Boolean</p>
exclusiveMode	<p>(Optional) Use one of the following options:</p> <ul style="list-style-type: none"> None - the "default", which does not retain a resource. Job - keeps the resource for the duration of the job. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" is used in a step. Future steps for this job will use this resource in preference to other resources--if this resource meets the needs of the steps and its step limit is not exceeded. Step - keeps the resource for the duration of the step. Call - keeps the resource for the duration of the procedure that called this step, which is equivalent to 'job' for top level steps. <p>Argument type: ExclusiveMode</p>
logFileName	<p>(Optional) A custom log file name produced by running the step. By default, ElectricFlow assigns a unique name for this file.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>parallel</code>	<p>(Optional) The value for <code>parallel</code> is a <i><Boolean flag - 0 1 true false></i>.</p> <p>If set, indicates this step should run at the same time as adjacent steps marked to run as parallel also. The default is to <code>false</code>.</p> <p>Argument type: Boolean</p>
<code>postProcessor</code>	<p>(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 <code>postProcessor</code> is a command string for invoking a post-processor program in the platform shell for the resource (<code>cmd</code> for Windows, <code>sh</code> for UNIX).</p> <p>Argument type: String</p>
<code>precondition</code>	<p>(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 <i>precondition</i> is evaluated.</p> <p>A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a <code>"0"</code> or <code>"false"</code> is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</p> <p>Precondition example: Assume we defined these 4 steps:</p> <ol style="list-style-type: none"> 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system <p>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.</p> <p>You can achieve optimal step execution order with preconditions:</p> <ul style="list-style-type: none"> • 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 (<code>/myJob/buildInstallerCompleted=1</code>). • Set a precondition in step 4: <code>\$/myJob/buildInstallerCompleted]</code> <p>Argument type: String</p>

Arguments	Descriptions
<code>releaseExclusive</code>	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>This declares whether or not this step will release its resource, which is currently held exclusively.</p> <p>Note: Setting this flag to "true" is the same as setting <code>releaseMode</code> to <code>release</code>.</p> <p>Argument type: Boolean</p>
<code>releaseMode</code>	<p>(Optional) Use one of the following options:</p> <ul style="list-style-type: none"> • <code>none</code> - the "default" - no action if the resource was not previously marked as "retain." • <code>release</code> - 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. • <code>releaseToJob</code> - allows a step to promote a "step exclusive" resource to a Job exclusive resource. <p>Argument type: ReleaseMode</p>
<code>resourceName</code>	<p>(Optional) Name for the resource that must be unique among all resources.</p> <p>Argument type: String</p>
<code>shell</code>	<p>(Optional) Where <i>shell</i> 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 (<code>cmd</code> for Windows, <code>sh</code> for UNIX). This is applicable to command steps only.</p> <p>Argument type: String</p>
<code>subprocedure</code>	<p>(Optional) The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the <code>command</code> or <code>commandFile</code> options.</p> <p>Argument type: String</p>
<code>subproject</code>	<p>(Optional) If a <code>subprocedure</code> argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>timeLimit</code>	(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
<code>timeLimitUnits</code>	(Optional) Specify <code>hours minutes seconds</code> for time limit units. Argument type: TimeLimitUnits
<code>workingDirectory</code>	(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
<code>workspaceName</code>	(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: `$cmdr->createStep(<projectName>, <procedureName>, <stepName>, {<optionals>});`

Specifying most arguments to the Perl `createStep` API is fairly intuitive. Similar to any other API, key-value pairs are specified in a hash argument for all optional parameters. However, specifying actual parameters is a little

different because they are not arbitrary key-values characterizing the step. Actual parameters are key-values

characterizing actual parameters to the step. See the following `createStep` request in XML:

```
<createStep>
  <projectName>MyProject</projectName>
  <procedureName>MyProcedure</procedureName>
  <stepName>Step1</stepName>
  <actualParameter>
    <actualParameterName>parm1</actualParameterName>
    <value>myval</value>
  </actualParameter>
  <actualParameter>
    <actualParameterName>parm2</actualParameterName>
```

```
        <value>val2</value>
    </actualParameter>
</createStep>
```

Each actual parameter key-value is under an `<actualParameter>` element, which is codified in the optional arguments hash in the Perl API like this:

```
{... => ..., actualParameter => [{actualParameterName => 'parm1', value =>
'myval'},
 {actualParameterName => 'parm2', value => 'val2'}], ... => ...}
```

In other words, the value of the `actualParameter` key in the optional arguments hash is a list of hashes, each representing one actual parameter. If the subprocedure call only takes one actual parameter, the value of the `actualParameter` key can be specified as just the hash representing the one parameter:

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

Example

```
$cmdr->createStep("Test Proj", "Run Build", "Common Cleanup", {subprocedure => "Delay",
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
<code>projectName</code>	The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String

Arguments	Descriptions
<code>procedureName</code>	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: `$cmdr->deleteProcedure(<projectName>, <procedureName>);`

Example

```
$cmdr->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
<code>projectName</code>	The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String
<code>procedureName</code>	The name of the procedure that must be unique within the project. Argument type: String
<code>stepName</code>	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: `$cmdr->deleteStep(<projectName>, <procedureName>, <stepName>);`

Example

```
$cmdr->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
<code>projectName</code>	The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String
<code>procedureName</code>	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: `$cmdr->getProcedure(<projectName>, <procedureName>);`

Example

```
$cmdr->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
<code>projectName</code>	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: `$cmdr->getProcedures(<projectName>);`

Example

```
$cmdr->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: \$cmdr->getStep(<projectName>, <procedureName>, <stepName>);

Example

```
$cmdr->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: `$cmdr->getSteps(<projectName>, <procedureName>);`

Example

```
$cmdr->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
<code>projectName</code>	<p>The name of the project that contains this procedure. The name must be unique among all projects. You must also enter <code>procedureName</code>.</p> <p>Argument type: String</p>
<code>procedureName</code>	<p>The name of the procedure that must be unique within the project. You must also enter <code>projectName</code>.</p> <p>Argument type: String</p>
<code>credentialName</code>	<p>(Optional) Name of the credential in one of these forms:</p> <ul style="list-style-type: none"> • relative (for example, <code>"cred1"</code>)—The credential is assumed to be in the project that contains the request target object. • absolute (for example, <code>"/projects/BuildProject/credentials/cred1"</code>)—The credential can be from any specified project, regardless of the target object's project. <p>Argument type: String</p>

Arguments	Descriptions
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
jobNameTemplate	<p>(Optional) Job name format for jobs created by running this procedure.</p> <p>Argument type: String</p>
newName	<p>(Optional) New name of the procedure.</p> <p>Argument type: String</p>
resourceName	<p>(Optional) The name of the default resource where steps belonging to this procedure will run. This name can be a resource pool name.</p> <p>Argument type: String</p>
timeLimit	<p>(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 <code>runProcedure</code> (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).</p> <p>Argument type: String</p>
timeLimitUnits	<p>(Optional) Time limit units are <code>hours minutes seconds</code>.</p> <p>Argument type: TimeLimitUnits</p>
workspaceName	<p>(Optional) The name of the default workspace where job output is stored.</p> <p>Argument type: String</p>

Positional arguments

projectName, procedureName

Response

Returns a modified procedure object.

ec-perl

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

Example

```
$cmdr->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
<code>projectName</code>	The name of the project that must be unique among all projects. You must also enter the <code>procedureName</code> . Argument type: String
<code>procedureName</code>	The name of the procedure that must be unique within the project. You must also enter <code>projectName</code> . Argument type: String
<code>stepName</code>	The name of the step that must be unique within the procedure. You must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>actualParameter</code>	(Optional) Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure. Argument type: Map
<code>afterProcedureStep</code>	(Optional) If specified, the procedure step will be placed after the named procedure step.. Argument type: String

Arguments	Descriptions
<code>alwaysRun</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If set to 1 or <code>true</code>, 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.</p> <p>Argument type: Boolean</p>
<code>beforeProcedureStep</code>	<p>(Optional) The procedure step will be placed before the named procedure step if this option is used.</p> <p>Argument type: String</p>
<code>broadcast</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i> 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 <code>resourceName</code>.</p> <p>Argument type: Boolean</p>
<code>clearActualParameters</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If set to <code>true</code> or 1, all the actual parameters will be removed from the step.</p> <p>Argument type: Boolean</p>
<code>command</code>	<p>(Optional) The step command.</p> <p>Argument type: String</p>
<code>commandFile</code>	<p>(Optional) This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. The contents of the <i>command file</i> is read and stored in the "command" field. This is an alternative argument for <code>command</code> and is useful if the "command" field spans multiple lines.</p>
<code>comment</code>	<p>(Optional) The script to execute the functions of this step, which are passed to the step's shell for execution.</p> <p>Argument type: String</p>
<code>condition</code>	<p>(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.</p> <p>Argument type: String</p>

Arguments	Descriptions
credentialName	<p>(Optional) Name of the credential in one of these forms:</p> <ul style="list-style-type: none"> • relative (for example, "cred1")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "projects/BuildProject/credentials/cred1")—The credential can be from any specified project, regardless of the target object's project. <p>Argument type: String</p>
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
errorHandling	<p>(Optional) Determines what happens to the procedure if the step fails:</p> <p><code>failProcedure</code> - The current procedure continues, but the overall status is error (default).</p> <ul style="list-style-type: none"> • <code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete. • <code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure. • <code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run. • <code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps. • <code>ignore</code> - Continues as if the step succeeded. <p>Argument type: ErrorHandling</p>
exclusive	<p>(Optional) <i><Boolean flag -0 1 true false></i>.</p> <p>If set to 1 or <code>true</code>, this indicates this step should acquire and retain this resource exclusively. The default is 0 or <code>false</code>.</p> <p>When you set <code>exclusive</code>, <code>exclusiveMode</code> is set to <code>job</code>.</p> <p>Argument type: Boolean</p>

Arguments	Descriptions
<code>exclusiveMode</code>	<p>(Optional) Use one of the following options:</p> <ul style="list-style-type: none"> • <code>None</code> - the "default", which does not retain a resource. • <code>Job</code> - 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. • <code>Step</code> - keeps the resource for the duration of the step. • <code>Call</code> - keeps the resource for the duration of the procedure that called this step, which is equivalent to 'job' for top level steps. <p>Argument type: <code>ExclusiveMode</code></p>
<code>logFileName</code>	<p>(Optional) A custom log file name produced by running the step. By default, ElectricFlow assigns a unique name to this file.</p> <p>Argument type: <code>String</code></p>
<code>newName</code>	<p>(Optional) New name of the step.</p> <p>Argument type: <code>String</code></p>
<code>parallel</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to <code>1</code> or <code>true</code>, this step and all adjacent steps marked with the flag set will run at the same time.</p> <p>Argument type: <code>String</code></p>
<code>postProcessor</code>	<p>(Optional) This command runs in parallel with the main command for the step. It analyzes the log for the step and collects diagnostic information.</p> <p>Argument type: <code>String</code></p>

Arguments	Descriptions
<code>precondition</code>	<p>(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 <i>precondition</i> is evaluated. A <i>precondition</i> 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.</p> <p>Precondition example: Assume we defined these 4 steps:</p> <ol style="list-style-type: none"> 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system <p>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.</p> <p>You can achieve optimal step execution order with preconditions:</p> <ul style="list-style-type: none"> • 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] <p>Argument type: String</p>
<code>postProcessor</code>	<p>(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.</p> <p>Argument type: String</p>
<code>releaseExclusive</code>	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>This declares whether or not this step will release its resource, which is currently held exclusively.</p> <p>Note: Setting this flag to 1 or true is the same as setting <code>releaseMode</code> to "release".</p> <p>Argument type: Boolean</p>

Arguments	Descriptions
<code>releaseMode</code>	<p>(Optional) Use one of the following options:</p> <ul style="list-style-type: none"> • <code>none</code> - the "default" - no action if the resource was not previously marked as "retain." • <code>release</code> - 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. • <code>releaseToJob</code> - allows a step to promote a Step exclusive resource to a Job exclusive resource. <p>Argument type: ReleaseMode</p>
<code>resourceName</code>	<p>(Optional) The name of the resource used by this step. It must be unique among all resources.</p> <p>Argument type: String</p>
<code>shell</code>	<p>(Optional) Where <i>shell</i> 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.</p> <p>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.</p> <p>Argument type: String</p>
<code>subprocedure</code>	<p>(Optional) The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the <code>command</code> or <code>commandField</code>.</p> <p>Argument type: String</p>
<code>subproject</code>	<p>(Optional) If a <code>subprocedure</code> argument is used, this is the name of the project where that subprocedure is found.</p> <p>By default, the current project is used.</p> <p>Argument type: String</p>
<code>timeLimit</code>	<p>(Optional) The maximum length of time the step is allowed to run. After the time specified, the step will be aborted.</p> <p>Argument type: String</p>
<code>timeLimitUnits</code>	<p>(Optional) Units for step time limit: <code><hours minutes seconds></code></p> <p>Argument type: TimeLimitUnits</p>

Arguments	Descriptions
<code>workingDirectory</code>	(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
<code>workspaceName</code>	(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: `$cmdr->modifyStep(<projectName>, <procedureName>, <stepName>, {<optionals>});`

Example

```
$cmdr->modifyStep("Default", "Run Build", "Compile", {commandFile => "tempfile.txt"});
```

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
<code>projectName</code>	The name of the project that contains this procedure. The name must be unique among all projects. You must also enter <code>procedureName</code> . Argument type: String

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure that must be unique within the project. You must also enter <code>projectName</code> . Argument type: String
<code>stepName</code>	The name of the step that must be unique within the procedure. You must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>beforeStep</code>	(Optional) Moves the step (<code>stepName</code>) to position before the step "named" by this option. If omitted, <code>stepName</code> 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: `$cmdr->moveStep(<projectName>, <procedureName>, <stepName>, {<optionals>});`

Example

```
$cmdr->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](#) on page 493

[deleteProcess](#) on page 494

[getProcess](#) on page 495

[getProcesses](#) on page 496

[modifyProcess](#) on page 497

[runProcess](#) on page 499

createProcess

Creates a new process for an application or component.

Required Arguments

`projectName`

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

Argument Type: String

`processName`

Description: Name of the process.

Argument Type: String

Optional Arguments

`applicationName`

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

`componentName`

Description: Name of the component, if the process is owned by a component.

Argument Type: String

`credentialName`

Description: Name of a credential to attach to this process.

Argument Type: String

`description`

Description: Comment text describing this object; not interpreted at all by 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:

```
$ec->createProcess("Default", "Deploy", {componentName => "VCscomponent", processType => DEPLOY});
```

ectool

Syntax:

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

Example:

```
ectool createProcess "Default"t "Deploy" --componentName "VCscomponent" --processType DEPLOY
```

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, if the process is owned by an application. It must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

```
$ec->deleteProcess("Default", "Deploy", {componentName => "Web Server"});
```

ectool

Syntax:

```
ectool deleteProcess <projectName> <processName> [optionals]
```

Example:

```
ectool deleteProcess "Default" "Deploy" --componentName "Web Server"
```

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, if the process is owned by an application, that must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Retrieves the specified process element.

ec-perl

Syntax:

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

Example:

```
$ec->getProcess("Default", "Deploy", {componentName => "VCS"});
```

ectool

Syntax:

```
ectool getProcess <projectName> <processName> [optionals]
```

Example:

```
ectool getProcess "Default" "Deploy" --componentName VCScomponent
```

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, if the process is owned by an application, that must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Retrieves the specified process element.

ec-perl

Syntax:

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

Example:

```
$ec->getProcesses("Default", {componentName => "VCS component"});
```

ectool

Syntax:

```
ectool getProcesses <projectName> [optionals]
```

Example:

```
ectool getProcesses "Default" --componentName "VCS component"
```

modifyProcess

Modifies an existing process.

Required Arguments

projectName

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

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

Optional Arguments

applicationName

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String`credentialName`**Description:** Name of a credential to attach to this process.**Argument Type:** String`description`**Description:** Comment text describing this object; not interpreted at all by 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**

Retrieves an updated process element.

ec-perl

Syntax:

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

Example:

```
$ec->modifyProcess("Default", "Deploy", {componentName => "VCS v1", newName => "VCS v2", description => "Updated component"}
```

ectool

Syntax:

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

Example:

```
ectool modifyProcess "Default" "Deploy" --componentName "VCS v1" --newName "VCS  
v2" --description "Updated component"
```

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`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 => "Main Librar
y"});
```

ectool

Syntax:

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

Example:

```
ectool runProcess "Default" "Deploy" "Check out" --tierMapName "Main Library"
```

API Commands - Process Dependency

[createProcessDependency](#) on page 502

[deleteProcessDependency](#) on page 503

[getProcessDependencies](#) on page 504

[modifyProcessDependency](#) on page 505

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, if the process is owned by an application. It must be unique among all projects.

Argument Type: String

`branchCondition`

Description: Condition of the branch.

Argument Type: String

`branchConditionName`

Description: Name of the branch condition.

Argument Type: String

`branchConditionType`

Description: Type of the branch condition.

Argument Type: BranchConditionType

`branchType`

Description: Type of the branch.

Argument Type: BranchType

`componentApplicationName`

Description: If specified, the component is scoped to this application not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Returns a process dependency element.

ec-perl

Syntax:

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

Example:

```
$ec->createProcessDependency("Default", "Deploy", "Get WAR file", "Copy WAR file",
    {componentName => "VCS component"});
```

ectool

Syntax:

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

Example:

```
ectool createProcessDependency "Default" "Deploy" "Get WAR file" "Copy WAR file"
    --componentName "VCS component"
```

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, if the process is owned by an application. It must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

```
$ec->deleteProcessDependency("Default", "Deploy", "Get WAR file", "Copy WAR file", {componentName => "VCS component"});
```

ectool

Syntax:

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

Example:

```
ectool deleteProcessDependency "Default" "Deploy" "Get WAR file" "Copy WAR file" --componentName "VCS component"
```

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, if the process is owned by an application; must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

`componentName`

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Retrieves zero or more process dependency elements.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

```
ectool getProcessDependencies "Default" "Undeploy" --componentName VCScomponent
```

modifyProcessDependency

Modifies a dependency between two process steps.

Required Arguments

`projectName`

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

Argument Type: String

processName

Description: Name of the process.

Argument Type: String

processStepName

Description: Name of the process step.

Argument Type: String

targetProcessStepName

Description: Name of the target process step.

Argument Type: String

Optional Arguments

applicationName

Description: Name of the application, if the process is owned by an application.

Argument Type: String

branchCondition

Description: Condition of the branch.

Argument Type: String

branchConditionName

Description: Name of the branch condition.

Argument Type: String

branchConditionType

Description: Type of the branch condition.

Argument Type: BranchConditionType

branchType

Description: Type of the branch.

Argument Type: BranchType

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

ec-perl

Syntax:

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

Example:

```
$ec->modifyProcessDependency("Default", "Deploy", "Retrieve", "Publish", {componentName => "VCS v1"});
```

ectool

Syntax:

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

Example:

```
ectool modifyProcessDependency "Default" "Deploy" "Retrieve" "Publish" --componentName "VCS v1"
```

API Commands - Process Step

[completeManualProcessStep](#) on page 507

[createProcessStep](#) on page 508

[deleteProcessStep](#) on page 512

[getProcessStep](#) on page 513

[getProcessSteps](#) on page 514

[modifyProcessStep](#) on page 515

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.

Argument Type: String

Response

None or status OK message.

ec-perl

Syntax:

```
$<object>->completeManualProcessStep(<jobStepId>, {<optionals>});
```

Example:

To approve the manual process step:

```
$ec->createProcessStep("4fa765dd-73f1-11e3-b67e-b0a420524153", {action => completed, comment => "Success"});
```

To reject the manual process step:

```
$ec->createProcessStep("4fa765de-73f1-11e3-b67e-b0a420524153", {action => failed, comment => "Try again"});
```

ectool

Syntax:

```
ectool createProcessStep <jobStepId> [optionals]
```

Example:

To approve the manual process step:

```
ectool createProcessStep "4fa765dd-73f1-11e3-b67e-b0a420524153" --action completed --comment "Success"
```

To reject the manual process step:

```
ectool createProcessStep "4fa765de-73f1-11e3-b67e-b0a420524153" --action failed --comment "Try again"
```

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, if the process is owned by an application; must be unique among all projects.

Argument Type: String

applicationTierName

Description: Application tier on which to run the step.

Argument Type: String

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 component scoped to the specified application, not to the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

credentialName

Description: Name of the credential object.

Argument Type: String

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:

```
$ec->createProcessStep("Default", "Deploy", "Get file", {componentName => "VCS component", instruction => "Restart server"});
```

ectool

Syntax:

```
ectool createProcessStep <projectName> <processName> <processStepName> [optional  
s]
```

Example:

```
ectool createProcessStep "Default" "Deploy" "Get file" --componentName "VCS comp  
onent" --instruction "Restart Server"
```

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, if the process is owned by an application; must be unique among all projects.

Argument Type: String

componentApplicationName

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

componentName

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:

```
$ec->deleteProcessStep ("Default", "Deploy", "Take Snapshot", {componentName=> "
VCS component"});
```

ectool

Syntax:

```
ectool deleteProcessStep <projectName> <processName> <processStepName> [optional
s]
```

Example:

```
ectool deleteProcessStep "Default" "Deploy" "Take Snapshot" --componentName "VCS
component"
```

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: The name of the process.

Argument Type: String

processStepName

Description: The name of the process step.

Argument Type: String

Optional Arguments

applicationEntityRevisionId

Description: The revision ID of the versioned object.

Argument Type: UUID

applicationName

Description: When you want to retrieve an application process step, specify the application that owns the application process step. The application name must be unique among all projects.

Argument Type: String

componentApplicationName

Description: When you want to retrieve a component process step, specify the component (componentName) and the application that owns the component (componentApplicationName).

Argument Type: String`componentName`

Description: When you want to retrieve a component process step, specify the component (`componentName`) and the application that owns the component (`componentApplicationName`).

Argument Type: String**Response**

Returns a process step element.

ec-perl

Syntax:

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

Example:

```
$ec->getProcessStep("Default", "Deploy", "Check out", {componentApplicationName => "Run", componentName => "WAR file"});
```

ectool

Syntax:

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

Example:

```
ectool getProcessStep "Default" "Deploy" "Check out" --componentApplicationName "Run" --componentName "WAR file"
```

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: The revision ID of the versioned project.

Argument Type: UUID`applicationName`

Description: Name of the application, if the process is owned by an application; must be unique among all projects.

Argument Type: String

`componentApplicationName`

Description: If specified, the component is scoped to this application, not the project.

Argument Type: String

`componentName`

Description: Name of the component, if the process is owned by a component.

Argument Type: String

Response

Retrieves zero or more process step elements.

ec-perl

Syntax:

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

Example:

```
$ec->getProcessSteps("Default", "Deploy", {componentName=> "VCscomponent"});
```

ectool

Syntax:

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

Example:

```
ectool getProcessSteps "Default" "Deploy" --componentName "VCscomponent"
```

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, if the process is owned by an application; it 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: If specified, the component is scoped to this application not the project.

Argument Type: String

`componentName`

Description: Name of the component, if the process is owned by a component.

Argument Type: String

`credentialName`

Description: Name of the credential object.

Argument Type: String

`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

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

Retrieves an updated process step element.

ec-perl

Syntax:

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

Example:


```
$ec->modifyProcessStep ("Default", "Deploy", "Retrieve", {componentName => "VCSc
omponent", newName => "Checkout",
description => "New process step"});
```

ectool

Syntax:

```
ectool modifyProcessStep <projectName> <processName> <processStepName> [optional
s]
```

Example:

```
ectool modify ProcessStep "Default", "Deploy", "Retrieve" --componentName "VCSc
omponent" --newName "Checkout"
--description "New process step"
```

API Commands - Project Management

[createProject](#) on page 519

[deleteProject](#) on page 520

[getProject](#) on page 521

[getProjects](#) on page 522

[modifyProject](#) on page 523

[reloadSetupScripts](#) on page 524

createProject

Creates a new project.

You must specify a `projectName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project that must be unique among all projects. Argument type: String
<code>credentialName</code>	(Optional) Name of the credential in one of these forms: <ul style="list-style-type: none"> • relative (for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the target object's project. Argument type: String

Arguments	Descriptions
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
resourceName	<p>(Optional) The name of the resource to use as the default for steps run by procedures in this project.</p> <p>Argument type: String</p>
tracked	<p>(Optional) <i><Boolean flag - 0 1 true false></i> If set to 1 or true, Change Tracking is enabled for the project.</p> <p>Argument type: Boolean</p>
workspaceName	<p>(Optional) The name of a workspace to use as the default for steps run by procedures in this project.</p> <p>Argument type: String</p>

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) <i><Boolean flag - 0 1 true false></i> 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
<code>projectName</code>	The name of the project that must be unique among all projects. Argument type: String
<code>credentialName</code>	(Optional) Name of the credential in one of these forms: <ul style="list-style-type: none"> • relative (for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "<i>/projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the target object's project. Argument type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String
<code>newName</code>	(Optional) New name of the project. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource used as the default for steps run by procedures in this project. Argument type: String
<code>tracked</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If set to 1 or true, Change Tracking is enabled for the project. Argument type: Boolean
<code>workspaceName</code>	(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

[createProperty](#) on page 525

[deleteProperty](#) on page 532

[expandString](#) on page 537

[getProperties](#) on page 549

[getProperty](#) on page 554

[incrementProperty](#) on page 560

[modifyProperty](#) on page 566

[setProperty](#) on page 572

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
<code>propertyName</code>	<p>The name of the property that must be unique within the property sheet.</p> <p>It can be a relative or absolute property path, including "my" paths such as <code>"/myProject/prop1"</code>.</p> <p>Argument type: String</p>
<code>applicationName</code>	<p>(Optional) The name of the application container of the property sheet which owns the property. The name must be unique among all projects.</p> <p>Argument type: String</p>
<code>applicationTierName</code>	<p>(Optional) The name of the application tier container of the property sheet which owns the property.</p> <p>Argument type: String</p>

Arguments	Descriptions
artifactName	(Optional) The name of the artifact container of the property sheet which owns the property. Argument type: String
artifactVersionName	(Optional) The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as " <code>groupId:artifactKey:version</code> " 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) <i><Boolean flag - 0 1 true false></i> 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: <ul style="list-style-type: none"> • relative (for example, "<code>cred1</code>")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>")—The credential can be from any specified project, regardless of the target object's project. Argument type: String
credentialProtected	(Optional) <i><Boolean flag - 0 1 true false></i> 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	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
environmentName	<p>(Optional) The name of the environment container of the property sheet that owns the property. The name must be unique among all projects.</p> <p>Argument type: String</p>
environmentTemplateName	<p>(Optional) The name of the environment template container of the property sheet that owns the property.</p> <p>Argument type: String</p>
environmentTemplateTierName	<p>(Optional) The name of the environment template tier container of the property sheet that owns the property.</p>
environmentTierName	<p>(Optional) The name of the environment tier container of the property sheet that owns the property.</p> <p>Argument type: String</p>
expandable	<p>(Optional) Whether or not the property is recursively expandable.</p> <p><Boolean flag - 0 1 true false></p> <p>This determines whether the property value will undergo property expansion when it is fetched. The default is <code>true</code>.</p> <p>Argument type: Boolean</p>
extendedContextSearch	<p>(Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property.</p> <p>Argument type: Boolean</p>
flowName	<p>(Optional) The name of the flow.</p> <p>Argument type: String</p>
flowRuntimeName	<p>(Optional) Name of the flow runtime.</p> <p>Argument Type: String</p>
flowRuntimeStateName	<p>(Optional) Name of the flow state.</p> <p>Argument Type: String</p>

Arguments	Descriptions
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>flowTransitionName</code>	(Optional) The name of the flow transition. Argument type: String
<code>gatewayName</code>	(Optional) The name of the gateway container of the property sheet. Argument type: String
<code>groupName</code>	(Optional) The name of the group container of the property sheet which owns the property. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: UUID
<code>objectId</code>	(Optional) The object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(Optional) Property path string. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	The name of the plugin container of the property sheet which owns the property. Argument type: String

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure container of the property sheet which owns the property. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process when the container is a process or process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the container is a process step. Argument type: String
<code>projectName</code>	(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
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
<code>propertyType</code>	(Optional) Type of property: <code><string sheet></code> This indicates whether to create a string property or a sub-sheet. The default is <code>string</code> . Argument type: PropertyType
<code>releaseName</code>	(Optional) The name of the Release container of the property sheet which owns the property. Argument type: String
<code>repositoryName</code>	(Optional) The name of the repository container of the property sheet which owns the property. Use the repository for artifact management. Argument type: String
<code>resourceName</code>	(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
<code>resourcePoolName</code>	(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 <code>projectName</code> . 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 <code>projectName</code> and <code>procedureName</code> . Argument type: String
systemObjectName	(Optional) The name of the special system object. In this context, only <code>server</code> 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 <code>propertyType</code>), 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 <i>valuefile</i> 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-73f1-11e3-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
<code>propertyName</code>	The name of the property that must be unique within the property sheet. Argument type: String
<code>applicationName</code>	(Optional) The name of the application that must be unique among all projects. Argument type: String
<code>applicationTierName</code>	(Optional) The name of the application tier. Argument type: String
<code>artifactName</code>	(Optional) The name of the artifact. Argument type: String
<code>artifactVersionName</code>	(Optional) The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "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
<code>componentName</code>	(Optional) The name of the component. Argument type: String

Arguments	Descriptions
<code>configName</code>	(Optional) The name of the email configuration. Argument type: String
<code>credentialName</code>	(Optional) Whether or not the property is recursively expandable. Name of the credential in one of these forms: <ul style="list-style-type: none"> • relative (for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "<i>projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the target object's project. Argument type: String
<code>environmentName</code>	(Optional) The name of the environment that must be unique among all projects. Argument type: String
<code>environmentTemplateName</code>	(Optional) The name of the environment template. Argument type: String
<code>environmentTemplateTierName</code>	(Optional) The name of the environment template tier. Argument type: String
<code>environmentTierName</code>	(Optional) The name of the environment tier. Argument type: String
<code>extendedContextSearch</code>	(Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property. <i><Boolean flag -0 1 true false></i> 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: <ol style="list-style-type: none"> 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 <code>true</code> . Argument type: Boolean
<code>flowName</code>	(Optional) The name of the flow. Argument type: String

Arguments	Descriptions
<code>flowRuntimeName</code>	(Optional) Name of the flow runtime. Argument Type: String
<code>flowRuntimeStateName</code>	(Optional) Name of the flow state. Argument Type: String
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>flowTransitionName</code>	(Optional) The name of the flow transition. Argument type: String
<code>gatewayName</code>	(Optional) The name of the gateway. Argument type: String
<code>groupName</code>	(Optional) The name of a group that contains this property. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: String
<code>objectId</code>	(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(Optional) Property path. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String

Arguments	Descriptions
<code>pluginName</code>	(Optional) The name of a plugin that may contain a property you want to delete. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure containing the property you want to delete. Also requires <code>projectName</code> Argument type: String
<code>processName</code>	(Optional) The name of the process when the container is a process or process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the container is a process step. Argument type: String
<code>projectName</code>	(Optional) The name of the project that contains the property you want to delete. Argument type: String
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet, assigned automatically when the property sheet is created. Argument type: UUID
<code>releaseName</code>	(Optional) The name of the Release. Argument type: String
<code>repositoryName</code>	(Optional) The name of the repository for artifact management. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource that contains the property you want to delete. Argument type: String
<code>resourcePoolName</code>	(Optional) The name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	(Optional) The name of the resource template. Argument type: String

Arguments	Descriptions
<code>scheduleName</code>	(Optional) The name of the schedule containing the property you want to delete. Also requires <code>projectName</code> . Argument type: String
<code>snapshotName</code>	(Optional) The name of the snapshot. Argument type: String
<code>stageName</code>	(Optional) The name of the stage. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) The name of the step containing the property you want to delete. Also requires <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>systemObjectName</code>	(Optional) The name of a special system object. Only "server" is valid in this context. Argument type: SystemObjectName
<code>taskName</code>	(Optional) The name of the task. Argument type: String
<code>transitionDefinitionName</code>	(Optional) The name of the transition definition. Argument type: String
<code>transitionName</code>	(Optional) The name of the transition. Argument type: String
<code>userName</code>	(Optional) The user name that contains this property. Argument type: String
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workflowName</code>	(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 Summary"
```

[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 <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> 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: <ul style="list-style-type: none"> • relative (for example, <code>"cred1"</code>)—The credential is assumed to be in the project that contains the request target object. • absolute (for example, <code>"/projects/BuildProject/credentials/cred1"</code>)—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

Arguments	Descriptions
<code>environmentTemplateTierName</code>	(Optional) Name of the environment template tier. Argument type: String
<code>environmentTierName</code>	(Optional) The name of the environment tier. Argument type: String
<code>flowName</code>	(Optional) The name of the flow. Argument type: String
<code>flowRuntimeName</code>	(Optional) Name of the flow runtime. Argument Type: String
<code>flowRuntimeStateName</code>	(Optional) Name of the flow state. Argument Type: String
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>flowTransitionName</code>	(Optional) The name of the flow transition. Argument type: String
<code>gatewayName</code>	(Optional) The name of the gateway. Argument type: String
<code>groupName</code>	(Optional) The name of a group where you might expand a string. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: String
<code>objectId</code>	(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . 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 <i>valuefile</i> is read and stored in the "value" field. This is an alternative argument for <i>value</i> 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
<code>credentialName</code>	<p>(Optional) The name of the credential containing the properties to retrieve.</p> <p>Name of the credential in one of these forms:</p> <ul style="list-style-type: none"> • relative (for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "<i>projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the target object's project. <p>Argument type: String</p>
<code>environmentName</code>	<p>(Optional) The name of the environment that must be unique among all projects.</p> <p>Argument type: String</p>
<code>environmentTemplateName</code>	<p>(Optional) Name of the environment template.</p> <p>Argument type: String</p>
<code>environmentTemplateTierName</code>	<p>(Optional) Name of the environment template tier.</p> <p>Argument type: String</p>
<code>environmentTierName</code>	<p>(Optional) The name of the environment tier.</p> <p>Argument type: String</p>
<code>expand</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>The default value is 1 (<i>true</i>), and the value of each property will be expanded.</p> <p>A value of 0 (<i>false</i>) will cause the unexpanded value of each property to be returned.</p> <p>Argument type: Boolean</p>
<code>filters</code>	<p>(Optional) A list of zero or more filter criteria definitions used to define objects to find.</p> <p>See the findObjects API for complete description for using filters.</p> <p>Argument type: Collection</p>
<code>flowName</code>	<p>(Optional) The name of the flow.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>flowRuntimeName</code>	(Optional) Name of the flow runtime. Argument Type: String
<code>flowRuntimeStateName</code>	(Optional) Name of the flow state. Argument Type: String
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>flowTransitionName</code>	(Optional) The name of the flow transition. Argument type: String
<code>gatewayName</code>	(Optional) The name of the gateway. Argument type: String
<code>groupName</code>	(Optional) The name of the group containing the properties to retrieve. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>maxIds</code>	(Optional) <code><id count></code> 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
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: String

Arguments	Descriptions
<code>numObjects</code>	<p>(Optional) <code><full object count></code> Specifies the number of full objects (not just the IDs) returned from the <code>findObjects</code> 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 <code>numObjects</code> is not specified, all full objects in the list of object IDs are returned. Any and all objects can be retrieved using the <code>getObjects</code> command.</p> <p>Argument type: Integer</p>
<code>objectId</code>	<p>(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</p> <p>Argument type: String</p>
<code>path</code>	<p>(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.</p> <p>Argument type: String</p>
<code>pipelineName</code>	<p>(Optional) The name of the pipeline.</p> <p>Argument type: String</p>
<code>pluginName</code>	<p>(Optional) The name of the plugin containing the properties to retrieve.</p> <p>Argument type: String</p>
<code>procedureName</code>	<p>The name of the procedure containing the properties to retrieve. Also requires <code>projectName</code>.</p> <p>Argument type: String</p>
<code>processName</code>	<p>(Optional) The name of the process when the container is a process or process step.</p> <p>Argument type: String</p>
<code>processStepName</code>	<p>(Optional) The name of the process step when the container is a process step.</p> <p>Argument type: String</p>
<code>projectName</code>	<p>(Optional) The name of the project containing the properties to retrieve.</p> <p>Argument type: String</p>
<code>releaseName</code>	<p>(Optional) The name of the Release.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>repositoryName</code>	(Optional) The name of the repository for artifact management. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource containing the properties to retrieve. Argument type: String
<code>resourcePoolName</code>	(Optional) The name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	(Optional) Name of the resource template. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule containing the properties to retrieve. Also requires <code>projectName</code> . Argument type: String
<code>snapshotName</code>	(Optional) The name of the snapshot. Argument type: String
<code>sorts</code>	(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
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(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 <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as " <code>groupId:artifactKey:version</code> " 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
<code>credentialName</code>	<p>(Optional) The name of the credential containing the properties to retrieve.</p> <p>Name of the credential in one of these forms:</p> <ul style="list-style-type: none"> • relative (for example, "<i>cred1</i>")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "<i>projects/BuildProject/credentials/cred1</i>")—The credential can be from any specified project, regardless of the target object's project. <p>Argument type: String</p>
<code>environmentName</code>	<p>(Optional) The name of the environment that must be unique among all projects.</p> <p>Argument type: String</p>
<code>environmentTemplateName</code>	<p>(Optional) Name of the environment template.</p> <p>Argument type: String</p>
<code>environmentTemplateTierName</code>	<p>(Optional) Name of the environment template tier.</p> <p>Argument type: String</p>
<code>environmentTierName</code>	<p>(Optional) The name of the environment tier.</p> <p>Argument type: String</p>
<code>expand</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>The default value is 1 (<i>true</i>), and the value of each property will be expanded.</p> <p>A value of 0 (<i>false</i>) will cause the unexpanded value of each property to be returned.</p> <p>Argument type: Boolean</p>
<code>flowName</code>	<p>(Optional) The name of the flow.</p> <p>Argument type: String</p>
<code>flowRuntimeName</code>	<p>(Optional) Name of the flow runtime.</p> <p>Argument Type: String</p>
<code>flowRuntimeStateName</code>	<p>(Optional) Name of the flow state.</p> <p>Argument Type: String</p>

Arguments	Descriptions
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>flowTransitionName</code>	(Optional) The name of the flow transition. Argument type: String
<code>gatewayName</code>	(Optional) The name of the gateway. Argument type: String
<code>groupName</code>	(Optional) The name of the group containing the properties to retrieve. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: String
<code>objectId</code>	(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(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
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(Optional) The name of the plugin containing the properties to retrieve. Argument type: String

Arguments	Descriptions
<code>procedureName</code>	The name of the procedure containing the properties to retrieve. Also requires <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process when the container is a process or process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the container is a process step. Argument type: String
<code>projectName</code>	(Optional) The name of the project containing the properties to retrieve. Argument type: String
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
<code>recurse</code>	(Optional) <i><Boolean flag - 0 1 true false></i> The default value is 0 (<i>false</i>), 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 <i>true</i> . Argument type: Boolean
<code>releaseName</code>	(Optional) The name of the release container of the property sheet which owns the property. Argument type: String
<code>repositoryName</code>	(Optional) The name of the repository for artifact management. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource containing the properties to retrieve. Argument type: String
<code>resourcePoolName</code>	(Optional) The name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	(Optional) Name of the resource template. Argument type: String

Arguments	Descriptions
<code>scheduleName</code>	(Optional) The name of the schedule containing the properties to retrieve. Also requires <code>projectName</code> . Argument type: String
<code>snapshotName</code>	(Optional) The name of the snapshot. Argument type: String
<code>stageName</code>	(Optional) The name of the stage. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) The name of the step containing the properties to retrieve. Also requires <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>systemObjectName</code>	(Optional) The name of the system object containing the properties to retrieve. Only "server" is supported. Argument type: SystemObjectName
<code>taskName</code>	(Optional) The name of the task. Argument type: String
<code>transitionDefinitionName</code>	(Optional) The name of the transition definition. Argument type: String
<code>transitionName</code>	(Optional) The name of the transition. Argument type: String
<code>userName</code>	(Optional) The name of the user containing the properties to retrieve. Argument type: String
<code>workflowDefinitionName</code>	(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	<p>The name or path for the property to retrieve. The name must be unique within the property sheet.</p> <div> IMPORTANT: Note: The names "properties" and "project" are not valid property names. </div> <p>Argument type: String</p>
applicationName	<p>(Optional) The name of the application that must be unique among all projects.</p> <p>Argument type: String</p>
applicationTierName	<p>(Optional) The name of the application tier.</p> <p>Argument type: String</p>
artifactName	<p>(Optional) The name of the artifact.</p> <p>Argument type: String</p>
artifactVersionName	<p>(Optional) The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly.</p> <p>Argument type: String</p>
componentName	<p>(Optional) The name of the component.</p> <p>Argument type: String</p>
configName	<p>(Optional) The name of the email configuration.</p> <p>Argument type: String</p>
credentialName	<p>(Optional) The name of the credential containing the property to retrieve.</p> <p>Name of the credential in one of these forms:</p> <ul style="list-style-type: none"> • relative (for example, "cred1")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "/projects/BuildProject/credentials/cred1")—The credential can be from any specified project, regardless of the target object's project. <p>Argument type: String</p>

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) <i><Boolean flag - 0 1 true false></i> 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. <i><Boolean flag - 0 1 true false></i> 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: <ul style="list-style-type: none"> • 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
<code>flowStateName</code>	(Optional) The name of the flow state. Argument type: String
<code>flowTransitionName</code>	(Optional) The name of the flow transition. Argument type: String
<code>gatewayName</code>	(Optional) The name of the gateway. Argument type: String
<code>groupName</code>	(Optional) The name of the group containing the property to retrieve. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: String
<code>objectId</code>	(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(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
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(Optional) The name of the plugin containing the property to retrieve. Argument type: String

Arguments	Descriptions
<code>procedureName</code>	(Optional) The name of the procedure containing the property to retrieve. Also requires <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process when the container is a process or process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the container is a process step. Argument type: String
<code>projectName</code>	(Optional) The name of the project containing the property to retrieve. Argument type: String
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
<code>releaseName</code>	(Optional) The name of the Release container of the property sheet which owns the property. Argument type: String
<code>repositoryName</code>	(Optional) The name of the repository for artifact management. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource containing the property to retrieve. Argument type: String
<code>resourcePoolName</code>	(Optional) The name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	(Optional) Name of the resource template. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule containing the property to retrieve. Also requires <code>projectName</code> . 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

Automatically 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 <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> 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: <ul style="list-style-type: none"> • relative (for example, <code>"cred1"</code>)—The credential is assumed to be in the project that contains the request target object. • absolute (for example, <code>"projects/BuildProject/credentials/cred1"</code>)—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

Arguments	Descriptions
environmentTierName	(Optional) The name of the environment tier. Argument type: String
extendedContextSearch	<p>(Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property.</p> <p><i><Boolean flag - 0 1 true false></i></p> <p>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:</p> <ul style="list-style-type: none"> • 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. <p>The default setting is true.</p> <p>Argument type: Boolean</p>
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
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: String
<code>objectId</code>	(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(Optional) Path to the property. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(Optional) The name of the plugin containing a property to increment.
<code>procedureName</code>	(Optional) The name of the procedure containing this property. Also requires <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process when the container is a process or process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the container is a process step. Argument type: String
<code>projectName</code>	The name of the project containing this property. Also requires <code>procedureName</code> . Argument type: String

Arguments	Descriptions
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
<code>releaseName</code>	(Optional) The name of the Release containing this property. Argument type: String
<code>repositoryName</code>	(Optional) The name of the repository for artifact management. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource containing this property. Argument type: String
<code>resourcePoolName</code>	The name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	Name of the resource template. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule containing this property. Also requires <code>projectName</code> . Argument type: String
<code>snapshotName</code>	(Optional) The name of the snapshot. Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) The name of the step containing this property. Also requires <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>systemObjectName</code>	(Optional) Only <code>server</code> 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 "Sample Project"
```

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

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

[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
<code>propertyName</code>	<p>The name of the property that must be unique within the property sheet.</p> <p>This argument can be a path.</p> <p>Argument type: String</p>
<code>applicationName</code>	<p>(Optional) The name of the application that must be unique among all projects.</p> <p>Argument type: String</p>
<code>applicationTierName</code>	<p>(Optional) The name of the application.</p> <p>Argument type: String</p>
<code>artifactName</code>	<p>(Optional) The name of the artifact.</p> <p>Argument type: String</p>
<code>artifactVersionName</code>	<p>(Optional) The name of the artifact version.</p> <p>Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name--the ElectricFlow server interprets either name form correctly.</p> <p>Argument type: String</p>

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) <i><Boolean flag -0 1 true false></i> 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) <i><Boolean flag - 0 1 true false></i> 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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a>
 <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr> </code> 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) <i><Boolean flag -0 1 true false></i> —Determines whether the property will undergo property expansion when it is retrieved. The default is <code>true</code> . Argument type: Boolean
extendedContextSearch	(Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property. <i><Boolean flag -0 1 true false></i> — 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: <ol style="list-style-type: none"> 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 <code>true</code> . 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 <code>findObjects</code> and <code>getObjects</code> . 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 <code>projectName</code> . 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) <code><string sheet></code> — Indicates whether to create a string property or a sub-sheet. The default is <code>string</code> . 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
<code>resourceName</code>	(Optional) Name of the resource. Argument type: String
<code>resourcePoolName</code>	(Optional) Name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	(Optional) Name of the resource template. Argument type: String
<code>scheduleName</code>	(Optional) Name of the schedule. Also requires <code>projectName</code> . Argument type: String
<code>snapshotName</code>	(Optional) The name of the snapshot. Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	(Optional) Name of the state definition. Argument type: String
<code>stateName</code>	(Optional) Name of the state. Argument type: String
<code>stepName</code>	Name of the step containing the property to be modified. Also requires <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>systemObjectName</code>	System objects include: <code>admin artifactVersions directory emailConfigs log plugins server session workspaces</code> . Argument type: <code>SystemObjectName</code>
<code>taskName</code>	(Optional) Name name of the task. Argument type: String
<code>transitionDefinitionName</code>	(Optional) Name of the transition definition. Argument type: String
<code>transitionName</code>	(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 <i>valuefile</i> is read and stored in the "value" field. This is an alternative argument for <i>value</i> 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
<code>propertyName</code>	The name or path of the property that must be unique within the property sheet. This argument can be a path. Argument type: String
<code>value</code>	The value of the property. Argument type: String
<code>valueFile</code>	This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the <i>valuefile</i> is read and stored in the "value" field. This is an alternative argument for <code>value</code> and is useful if the value field spans multiple lines.
<code>applicationName</code>	(Optional) The name of the application that must be unique among all projects. Argument type: String
<code>applicationTierName</code>	(Optional) The name of the application tier. Argument type: String
<code>artifactName</code>	(Optional) The name of the artifact. Argument type: String
<code>artifactVersionName</code>	(Optional) The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as "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
<code>componentName</code>	(Optional) The name of the component. Argument type: String

Arguments	Descriptions
configName	(Optional) The name of the email configuration. Argument type: String
counter	(Optional) <i><Boolean flag - 0 1 true false></i> 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: <ul style="list-style-type: none"> • relative (for example, "cred1")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "/projects/BuildProject/credentials/cred1")—The credential can be from any specified project, regardless of the target object's project. Also requires projectName. Argument type: String
description	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> 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) <i><Boolean flag - 0 1 true false></i> 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) <i><Boolean flag - 0 1 true false></i> 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
<code>groupName</code>	(Optional) The name of the group containing the property you want to set. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(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
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: String
<code>objectId</code>	(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(Optional) The path to the property. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(Optional) The name of the plugin containing the property you want to set. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure containing the property that you want to set. Also requires <code>projectName</code> .(Optional) Argument type: String
<code>processName</code>	(Optional) The name of the process when the container is a process or process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the container is a process step. Argument type: String

Arguments	Descriptions
<code>projectName</code>	(Optional) The name of the project containing the property that you want to set. Argument type: String
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: String
<code>releaseName</code>	(Optional) The name of the Release. Argument type: String
<code>repositoryName</code>	(Optional) The name of the repository for artifact management. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource containing the property that you want to set. Argument type: String
<code>resourcePoolName</code>	(Optional) The name of a pool containing one or more resources. Argument type: String
<code>resourceTemplateName</code>	(Optional) Name of the resource template. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule containing the property you want to set. Also requires <code>projectName</code> . Argument type: String
<code>snapshotName</code>	(Optional) The name of the snapshot. Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(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 Project"});
```

```
$cmdr->setProperty("/myResource/Application Path", "c:\Program Files\Application");
```

```
$cmdr->setProperty("Application Path", "c:\Program Files\Application",  
    {resourceName => "r2"});
```

ectool

syntax: ectool setProperty <propertyName> [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](#) on page 579

[addResourceToEnvironmentTier](#) on page 579

[createResource](#) on page 580

[createResourcePool](#) on page 584

[deleteResource](#) on page 585

[deleteResourcePool](#) on page 586

[getResource](#) on page 587

[getResources](#) on page 587

[getResourcesInEnvironmentTier](#) on page 588

[getResourcesInPool](#) on page 589

[getResourcePool](#) on page 590

[getResourcePools](#) on page 590

[getResourceUsage](#) on page 591

[modifyResource](#) on page 591

[modifyResourcePool](#) on page 595

[pingAllResources](#) on page 596

[pingResource](#) on page 597

[removeResourceFromEnvironmentTier](#) on page 597

[removeResourcesFromPool](#) on page 598

[signCertificate](#) on page 599

addResourcesToPool

Adds resources to the specified resource pool (a named group of resources).

You must specify a `resourcePoolName`.

Arguments	Descriptions
<code>resourcePoolName</code>	<p>Name for the resource pool that must be unique among all resource pools.</p> <p>The resource pool has one or more resources.</p> <p>Argument Type: String</p>
<code>resourceNames</code>	<p>(Optional) List of resources to add to the resource pool.</p> <p>Argument Type: Collection</p>

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
<code>resourceName</code>	Name for the resource that must be unique among all resources. Argument Type: String
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>environmentName</code>	Name of the environment that must be unique among all projects. Argument Type: String
<code>environmentTierName</code>	Name for the environment tier that must be unique among all tiers for the environment. 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>);
```

Example:

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

ectool

Syntax:

```
addResourceToEnvironmentTier <resourceName> <projectName> <environmentName> <env  
ironmentTierName>
```

Example:

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

[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
<code>resourceName</code>	Name of the new resource that must be unique among all resources. Argument Type: String
<code>artifactCacheDirectory</code>	(Optional) Directory on the agent host where retrieved artifacts are stored. Argument Type: String
<code>block</code>	(Optional) <i><Boolean flag - 0 1 true false></i> A newly created resource will be pinged. This argument makes the <code>createResource</code> call block until the result of the ping is known. The default is <i>false</i> . Argument Type: Boolean
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument Type: String
<code>hostName</code>	(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
<code>hostType</code>	(Optional) Type of the host. Argument Type: String
<code>pools</code>	(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
<code>port</code>	<p>(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.</p> <p>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 <code>ssh</code>, the default is 22.</p> <p>Argument Type: Integer</p>
<code>proxyCustomization</code>	<p>(Optional) Customized Perl code specifying how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources.</p> <p>Argument Type: String</p>
<code>proxyHostName</code>	<p>(Optional) The name or IP address of the computer containing the ElectricFlow Agent used for a proxy resource.</p> <p>Argument Type: String</p>
<code>proxyPort</code>	<p>(Optional) The ElectricFlow agent port number for a proxy resource. See the <code>port</code> argument description for more details.</p> <p>Argument Type: Integer</p>
<code>proxyProtocol</code>	<p>(Optional) Protocol for communicating with the proxy target. Defaults to <code>ssh</code>. (This argument is not exposed in the ElectricFlow UI at this time.)</p> <p>Argument Type: String</p>
<code>repositoryNames</code>	<p>(Optional) A list of one or more repository names. Each repository name is listed on a "new line."</p> <p>Argument Type: String</p>
<code>resourceDisabled</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this is set to 1, ElectricFlow will not start new steps on this resource. The default is <code>false</code>.</p> <p>Argument Type: Boolean</p>
<code>shell</code>	<p>(Optional) This sets a default shell for running step commands on this resource.</p> <p>The default is <code>"cmd /q /c"</code> for a Windows agent and <code>"sh -e"</code> for a UNIX agent.</p> <p>Argument Type: String</p>

Arguments	Descriptions
<code>stepLimit</code>	<p>(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.</p> <p>Argument Type: Integer</p>
<code>trusted</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this is set to 1 or <code>true</code>, the resource is <i>trusted</i>.</p> <p>Agents can be either <i>trusted</i> or <i>untrusted</i>:</p> <ul style="list-style-type: none"> • 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. <p>Argument Type: Boolean</p>
<code>useSSL</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>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 <code>true</code>.</p> <p>Note: Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server.</p> <p>Argument Type: Boolean</p>
<code>workspaceName</code>	<p>(Optional) Name of the workspace that the resource uses.</p> <p>Argument Type: String</p>
<code>zoneName</code>	<p>(Optional) The name of the zone where this resource resides.</p> <p>Argument Type: String</p>

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 P2"});
```

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
<code>resourcePoolName</code>	Name for the resource pool that must be unique among all resource pools. Argument Type: String
<code>autoDelete</code>	(Optional) <i><Boolean flag - 0 1 true false></i> – 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
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument Type: String
<code>orderingFilter</code>	(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
<code>resourceNames</code>	(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
<code>resourcePoolDisabled</code>	(Optional) <i><Boolean flag - 0 1 true false></i> –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", "Machine2"]});
```

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
<code>resourceName</code>	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
<code>resourcePoolName</code>	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
<code>resourceName</code>	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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>environmentName</code>	Name of the environment that must be unique among all projects. Argument Type: String
<code>environmentTierName</code>	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>, <environmentTierName>);
```

Example:

```
$ec->getResourcesInEnvironmentTier("Default", "PROD", "Tomcat");
```

ectool

Syntax:

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

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
<code>resourcePoolName</code>	The name of a resource pool containing one or more resources. Argument Type: String
<code>jobStepId</code>	(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
<code>resourcePoolName</code>	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
<code>resourceName</code>	Name for the resource that must be unique among all resources. Argument Type: String
<code>artifactCacheDirectory</code>	(Optional) The directory on the agent host where retrieved artifacts are stored. Argument Type: String
<code>block</code>	(Optional) <i><Boolean flag - 0 1 true false></i> A newly modified resource will be pinged. This argument makes the <code>modifyResource</code> call "block" until the result of the ping is known. The default is "false". Argument Type: Boolean
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument Type: String
<code>hostName</code>	(Optional) The name or IP address for the ElectricFlow machine containing the agent for this resource. Argument Type: String
<code>hostType</code>	(Optional) The type of the host. Argument Type: String
<code>newName</code>	(Optional) New name of an existing resource. Argument Type: String
<code>pools</code>	(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
<code>port</code>	<p>(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.</p> <p>Argument Type: Integer</p>
<code>proxyCustomization</code>	<p>(Optional) Customized Perl code specifying how the proxy resource communicates with the proxy target. This applies only to proxy resources.</p> <p>Argument Type: String</p>
<code>proxyHostName</code>	<p>(Optional) The IP address of the computer containing the ElectricFlow Agent used for a proxy resource.</p> <p>Argument Type: String</p>
<code>proxyPort</code>	<p>(Optional) The ElectricFlow agent port number for a proxy resource. See the <code>port</code> argument for more details.</p> <p>Argument Type: Integer</p>
<code>proxyProtocol</code>	<p>(Optional) Protocol for communicating with the proxy target. Defaults to <code>ssh</code>. This argument is not exposed in the ElectricFlow UI at this time.</p> <p>Argument Type: String</p>
<code>repositoryNames</code>	<p>(Optional) A list of repository names with each repository name listed on a "new line".</p> <p>Argument Type: String</p>
<code>resourceDisabled</code>	<p><Boolean flag - 0 1 true false> If this is set to 1 or true, ElectricFlow will not start new steps on this resource.</p> <p>Argument Type: String</p>
<code>shell</code>	<p>(Optional) This sets a default shell for running step commands on this resource. The default is "<code>cmd /q /c</code>" for a Windows agent and "<code>sh -e</code>" for a UNIX agent.</p> <p>Argument Type: String</p>
<code>stepLimit</code>	<p>(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.</p> <p>Argument Type: String</p>

Arguments	Descriptions
trusted	<p>(Optional) <i><Boolean flag - 0 1 true false></i> If this is set to 1 or true, the resource is <i>trusted</i>.</p> <p>Agents can be either <i>trusted</i> or <i>untrusted</i>:</p> <ul style="list-style-type: none"> • 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. <p>Argument Type: Boolean</p>
useSSL	<p>(Optional) <i><Boolean flag - 0 1 true false></i> 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.</p> <p>The default is 1 or true.</p> <p>Note: Transport Layer Security (TLS) has replaced Secure Sockets Layer version 3.0 (SSLv3) on the ElectricFlow web server and the ElectricFlow server.</p> <p>Argument Type: Boolean</p>
workspaceName	<p>(Optional) Name of the default workspace where job output is stored.</p> <p>Argument Type: String</p>
zoneName	<p>(Optional) Name of the zone where this resource resides, which must be unique among all zones.</p> <p>Argument Type: String</p>

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
<code>resourcePoolName</code>	Name for the resource pool that must be unique among all resource pools. A resource pool contains one or more resources. Argument Type: String
<code>autoDelete</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If this is set to 1 or <code>true</code> , the resource pool will be deleted automatically when the last resource is removed from the pool. Argument Type: Boolean
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument Type: String
<code>newName</code>	(Optional) Any new unique name for this resource pool. Argument Type: String
<code>orderingFilter</code>	(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
<code>resourceNames</code>	(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
<code>resourcePoolDisabled</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If this is set 1 or <code>true</code> , 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
<code>resourceName</code>	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
<code>resourceName</code>	Name for the resource that must be unique among all resources. Argument Type: String

Arguments	Descriptions
<code>projectName</code>	Name for the project that must be unique among all projects. Argument Type: String
<code>environmentName</code>	Name of the environment that must be unique among all projects. Argument Type: String
<code>environmentTierName</code>	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
<code>resourcePoolName</code>	Name for the resource pool that must be unique among all resource pools. Argument Type: String

Arguments	Descriptions
<code>resourceNames</code>	(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", "Tester 2", "Tester3"]});
```

ectool

syntax: `ectool removeResourcesFromPool <resourcePoolName> (optionals)`

Example

```
ectool removeResourcesFromPool "Test machines" --resourceNames "Tester 1" "Tester 2" "Tester 3"
```

[Back to Top](#)

signCertificate

Signs an agent certificate.

Arguments	Descriptions
<code>certificateData</code>	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 ("MIIIEczCCA1ugAwIBAgIBADANBgqhkiG9w0BAQQFAD");
```

ectool

syntax: `ectool signCertificate <certificateData>`

Example

```
ectool signCertificate "MIIIEczCCA1ugAwIBAgIBADANBgqhkiG9w0BAQQFAD"
```

[Back to Top](#)

API Commands - Schedule Management

[createSchedule](#) on page 600

[deleteSchedule](#) on page 604

[getSchedule](#) on page 605

[getSchedules](#) on page 606

[modifySchedule](#) on page 607

[pauseScheduler](#) on page 611

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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument type: String
<code>scheduleName</code>	Name for the schedule that must be unique among all schedules for the project. Argument type: String
<code>actualParameters</code>	(Optional) Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure. Argument type: Map

Arguments	Descriptions
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. <i>credentialName</i> 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> <div> <dl> <i> <p> <pre> <style> <table> <tc> <td> <th> <tr> 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
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

Arguments	Descriptions
<code>environmentTemplateProjectName</code>	(Optional) Name of the project containing specified environment template. Argument Type: String
<code>environmentTemplateTierMapName</code>	(Optional) The name of the environment template tier map used to create the dynamic environment where the application runs. Argument type: String
<code>interval</code>	(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
<code>intervalUnits</code>	(Optional) Specifies the units for the <code>interval</code> argument <code><hours minutes seconds continuous></code> If set to <code>continuous</code> , ElectricFlow creates a new job as soon as the previous job completes. Argument type: String
<code>misfirePolicy</code>	(Optional) <code><ignore runOnce></code> 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
<code>monthDays</code>	(Optional) Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space. Argument type: String
<code>priority</code>	(Optional) <code><low normal high highest></code> 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

Arguments	Descriptions
<code>procedureName</code>	(Optional) Name of the procedure to run when the schedule is invoked. Argument type: String
<code>processName</code>	(Optional) Name of the application process to run when the schedule is invoked. Argument type: String
<code>scheduleDisabled</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If set to 1, ElectricFlow will not start any new jobs from the schedule. Defaults to "false". Argument type: Boolean
<code>snapshotName</code>	(Optional) Name of the snapshot used to invoke the application process. Argument type: String
<code>startTime</code>	(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 <code>hh:mm</code> , using the 24-hour clock (for example, 17:00). Argument type: String
<code>startingStateName</code>	(Optional) Name of the starting state of the workflow. Argument type: String
<code>stopTime</code>	(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 <code>stopTime</code> is not specified, ElectricFlow creates one job only on each specified day. Enter hours and minutes, formatted <code>hh:mm</code> , using the 24-hour clock (for example, 17:00). Argument type: String
<code>tierMapName</code>	(Optional) Name of the tier map that determines in which environment the application runs. Argument type: String
<code>tierResourceCounts</code>	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
<code>projectName</code>	The name for the project that must be unique among all projects. Argument type: String
<code>applicationName</code>	(Optional) The name of the application that owns the process. Argument type: String
<code>applicationProjectName</code>	(Optional) The name for the project to which the application belongs. Argument type: String
<code>includeWorkflows</code>	(Optional) To include workflow related schedules. Argument type: String
<code>processName</code>	(Optional) The name of the application process. Argument type: String
<code>tierMapName</code>	(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
<code>projectName</code>	The name of the project that must be unique among all projects. Argument type: String
<code>scheduleName</code>	The name of the schedule that must be unique among all schedules for the project. Argument type: String
<code>actualParameters</code>	(Optional) Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure. Argument type: Map
<code>applicationProjectName</code>	(Optional) Name of the project to which the application belongs. Argument type: String
<code>beginDate</code>	(Optional) <code><yyyy-mm-dd></code> The date you want the schedule to begin (for example, 2016-05-15). Argument type: String

Arguments	Descriptions
<code>clearActualParameters</code>	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>If set to <code>true</code> or <code>1</code>, all the actual parameters are removed from the schedule.</p> <p>Argument type: Boolean</p>
<code>credentialName</code>	<p>(Optional) The name of the credential to use for user impersonation when running the procedure.</p> <p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object.</p> <p>absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project.</p> <p>Argument type: String</p>
<code>description</code>	<p>(Optional) A plain text or HTML description for this object.</p> <p>If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
<code>endDate</code>	<p>(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.</p> <p>Argument type: String</p>
<code>environmentName</code>	<p>(Optional) The name of the environment where the process will run.</p> <p>Argument type: String</p>
<code>environmentProjectName</code>	<p>(Optional) The name of the project to which the environment or environment template belongs.</p> <p>Argument type: String</p>
<code>environmentTemplateName</code>	<p>(Optional) The name of the environment template used to determine where the process will run.</p> <p>Argument type: String</p>
<code>environmentTemplateName</code>	<p>(Optional) Name of the project containing specified environment template. If this argument is not specified, the default is the environment project name.</p> <p>Argument Type: String</p>

Arguments	Descriptions
environmentTemplateTierMapName	(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
interval	(Optional) Determines the repeat interval for starting new jobs. Argument type: String
intervalUnits	(Optional) Specifies the units for the <code>interval</code> argument <code><hours minutes seconds continuous></code> . If set to <code>continuous</code> , ElectricFlow creates a new job as soon as the previous job completes. Argument type: IntervalUnits
misfirePolicy	(Optional) <code><ignore runOnce></code> —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 <code>ignore</code> . Argument type: IntervalUnits
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
priority	(Optional) <code><low normal high highest></code> 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

Arguments	Descriptions
processName	(Optional) The name of the process to run when the schedule is invoked. Argument type: String
scheduleDisabled	(Optional) <i><Boolean flag - 0 1 true false></i> 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 <i>hh:mm</i> , using the 24-hour clock (for example, 17:00). ElectricFlow starts creating jobs at this time on the days specified. 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 <i>stopTime</i> is not specified, ElectricFlow creates one job only on each specified day. Enter hours and minutes, formatted <i>hh:mm</i> , 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
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

Arguments	Descriptions
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 - Server Management

[deleteLicense](#) on page 612

[getAdminLicense](#) on page 613

[getCertificates](#) on page 614

[getLicense](#) on page 615

[getLicenses](#) on page 616

[getLicenseUsage](#) on page 616

[getServerInfo](#) on page 617

[getServerStatus](#) on page 617

[getVersions](#) on page 619

[importLicenseData](#) on page 620

[logMessage](#) on page 620

[setLogLevel](#) on page 621

[shutdownServer](#) on page 622

[tunePerformance](#) on page 623

deleteLicense

Deletes a license.

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

Arguments	Descriptions
<code>productName</code>	The name of the product with the licensed feature. Possible products include: <code>ElectricFlow</code> . Argument type: String

Arguments	Descriptions
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
<code>featureName</code>	The name of the licensed feature. Possible features include: <code>Server</code> Argument: String
<code>productName</code>	The name of the product with the licensed feature. Possible products include: <code>ElectricFlow</code> 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	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If the argument is set to 1 or true, the system waits until the server reaches a <i>terminal state</i>. Terminal states include running, failed, stopping, and importFailed.</p> <p>Argument: Boolean</p>
diagnostics	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>Select the diagnostic information that you want in your output:</p> <ul style="list-style-type: none"> • 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. <p>Argument: Boolean</p>
serverStateOnly	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>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.</p> <p>Argument: Boolean</p>
timeout	<p>(Optional) This argument specifies the timeout for the element flag.</p> <p>The default value is 120 seconds.</p> <p>Argument: Integer</p>

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
<code>licenseData</code>	The content of a license file (perl XML API).
<code>licenseFile</code>	<localFileName> The license file to import. This is a local file that will be read by ectool. The contents is sent as the <code>licenseData</code> 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
<code>message</code>	Message to add to the server log. Argument: String
<code>level</code>	(Optional) Select a message level: TRACE, DEBUG, INFO, WARN, ERROR, or OFF. Argument: String

Arguments	Descriptions
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
<code>force</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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
<code>restart</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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
<code>apiQueueSize</code>	Maximum number of threads that will be created in the API thread pool. Argument: Integer
<code>dbConnectionPoolSize</code>	Maximum number of database connections that the server will use. Argument: Integer
<code>dbThreadPoolSize</code>	Number of worker threads in the database connection manager. Argument: Integer
<code>dispatchQueueSize</code>	Maximum number of threads that will be created in the Dispatch thread pool. Argument: Integer
<code>quartzQueueSize</code>	Maximum number of threads that will be created in the Quartz thread pool. Argument: Integer
<code>stateMachineQueueSize</code>	Maximum number of threads that will be created in the <code>stateMachine</code> 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

[createSnapshot](#) on page 624

[deleteSnapshot](#) on page 626

[getPartialApplicationRevision](#) on page 626

[getSnapshot](#) on page 627

[getSnapshotEnvironments](#) on page 628

[getSnapshots](#) on page 629

[modifySnapshot](#) on page 630

createSnapshot

Creates a new snapshot of the specified application.

You must specify `projectName` , `applicationName`, and `snapshotName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project that must be unique among all projects. Argument type: String
<code>applicationName</code>	The name of the application that must be unique among all projects. Argument type: String
<code>snapshotName</code>	The name of the snapshot that must be unique withing the application. Argument type: String
<code>componentVersions</code>	(Optional) The name and version of the component for the snapshot. Argument type: Map

Arguments	Descriptions
description	<p>(Optional) Comment text describing this object, which is not interpreted by ElectricFlow.</p> <p>A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
environmentName	<p>(Optional) Name of the environment from which the snapshot is created.</p> <p>Argument type: String</p>
environmentProjectName	<p>(Optional) Name of the project to which the snapshot belongs.</p> <p>Argument type: String</p>
environmentTemplateProjectName	<p>(Optional) Name of the project to which the environment template belongs when the snapshot is created from the environment template.</p> <p>Argument type: String</p>

Positional arguments

projectName, applicationName, snapshotName

Response

Returns a snapshot element.

ec-perl

syntax: \$cmdr->createSnapshot (<projectName>, <applicationName>, <snapshotName>, {<optionals>});

Example

```
$cmdr->createSnapshot ("Default", "Deploy", "Test Run 1" {description => "Test use cases"});
```

ectool

syntax:]ectool createSnapshot <projectName> <applicationName> <snapshotName> [optionals]

Example

```
ectool createSnapshot "Default" "Deploy" "Test Run 1" --description "Test use cases"
```

[Back to Top](#)

deleteSnapshot

Deletes snapshot from an application.

You must specify `projectName`, `applicationName`, and `snapshotName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project that must be unique among all projects. Argument type: String
<code>applicationName</code>	The name of the application that must be unique among all projects. Argument type: String
<code>snapshotName</code>	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: `$cmdr->deleteSnapshot (<projectName>, <applicationName>, <snapshotName>);`

Example

```
$cmdr->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: \$cmdr->getPartialApplicationRevision (<projectName>, <applicationName>, <revisionNumber>);

Example

```
$cmdr->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

Arguments	Descriptions
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: \$cmdr->getSnapshot (<projectName>, <applicationName>, <snapshotName>);

Example

```
$cmdr->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: \$cmdr->getSnapshotEnvironments (<projectName>, <applicationName>, <snapshotName>);

Example

```
$cmdr->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: \$cmdr->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 `projectName`, `applicationName`, and `snapshotName`.

Arguments	Descriptions
<code>projectName</code>	The name of the project that must be unique among all projects. Argument type: String
<code>applicationName</code>	The name of the application that must be unique within the project. Argument type: String
<code>snapshotName</code>	The name of the snapshot that must be unique within the application. Argument type: String
<code>componentVersions</code>	(Optional) Name and version of the component for the snapshot. Use keyword <code>LATEST</code> for the latest version. Argument type: Map
<code>description</code>	(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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String
<code>environmentName</code>	(Optional) Name of environment from which snapshot is created. Argument type: String
<code>environmentProjectName</code>	(Optional) Name for the project to which the environment or environment template belongs. Argument type: String

Arguments	Descriptions
newName	(Optional) New name of the snapshot. Argument type: String

Positional arguments

projectName, applicationName, snapshot

Response

Returns a modified snapshot object.

ec-perl

syntax: \$cmdr->createSnapshot (<projectName>, <applicationName>, <snapshotName>, {<optionals>});

Example

```
$cmdr->createSnapshot ("Default", "Deploy", "Beta Version 4", {description => "Internal 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](#) on page 631

[deleteTierMap](#) on page 633

[deleteTierMapping](#) on page 634

[getTierMaps](#) on page 635

[modifyTierMap](#) on page 635

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 Apps"
--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>, <environmentProjectName>, <environmentName>);
```

Example:

```
$ec->deleteTierMap("Default", "Deploy", "Q4 Summary", "App Server");
```

ectool

Syntax:

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

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>, <environmentProjectName>, <environmentName>, <applicationTierName>);
```

Example:

```
$ec->deleteTierMap("Default", "Deploy", "Q4 Summary", "PROD", "Config");
```

ectool

Syntax:

```
ectool deleteTierMapping <projectName> <applicationName> <environmentProjectName> <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 => "4fa76a765dd-73f1-11e3-b67e-b0a420524153"});
```

ectool

Syntax:

```
ectool getTierMaps <projectName> <applicationName> [optionals]
```

Example:

```
ectool getTierMaps "Default" "Take snapshot" --applicationEntityRevisionId 4fa765dd-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>, <environmentProjectName>, <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](#) on page 637

[createGroup](#) on page 638

[createUser](#) on page 639

[deleteGroup](#) on page 641

[deleteUser](#) on page 641

[getGroup](#) on page 642

[getGroups](#) on page 643

[getUser](#) on page 644

[getUsers](#) on page 644

[login](#) on page 646

[logout](#) on page 646

[modifyGroup](#) on page 647

[modifyUser](#) on page 649

[removeUsersFromGroup](#) on page 650

addUsersToGroup

Adds one or more specified users to a local group.

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

Arguments	Descriptions
<code>groupName</code>	The name of the group that must be unique among local groups.
<code>userName</code>	Using ec-perl, enter one or more user names to add to the group.

Arguments	Descriptions
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

Arguments	Descriptions
<code>userName</code>	(Optional) Using ectool, enter one user name to add to the group. Argument type: String
<code>userNames</code>	(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", "cclark"]});
```

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
<code>userName</code>	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
<code>email</code>	(Optional) The email address of the user. Argument type: String
<code>fullUserName</code>	(Optional) Full name of the user, not a nickname. Argument type: String
<code>groupNames</code>	(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
<code>password</code>	(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
<code>groupName</code>	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

Positional arguments

groupName

Response

One [group](#) element.

ec-perl

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

Example

```
$cmdr->getGroup("QA group", {providerName => "LDAP"});
```

ectool

syntax: `ectool getGroup <groupName> [optionals]`

Example

```
ectool getGroup "QA group" --providerName "LDAP"
```

[Back to Top](#)

getGroups

Retrieves all groups.

Arguments	Descriptions
<code>filter</code>	(Optional) A string used to filter the returned groups by their names. Argument type: String
<code>includeAll</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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
<code>maximum</code>	(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
<code>userName</code>	The name of the user. Argument type: String
<code>providerName</code>	(Optional) The name of the directory provider, which limits the search to the specified directory provider. Argument type: String

Positional arguments

`userName`

Response

One `user` element.

ec-perl

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

Example

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

ectool

syntax: `ectool getUser <userName> [optionals]`

Example

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

[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
<code>filter</code>	(Optional) <i><filter pattern></i> Enter a filter pattern to match user names. The filter is not case sensitive and can include the "*" wildcard character. Argument type: String
<code>includeAll</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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
<code>maximum</code>	(Optional) <i><number of users></i> Specify a larger number of user records to retrieve. The default limit is 100 user records. Argument type: Integer
<code>searchFields</code>	(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
<code>password</code>	The password for the user who is "logging in". Argument type: String
<code>userName</code>	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
<code>groupName</code>	Name of the group that must be unique among local groups. Argument type: String
<code>migrateSettings</code>	(Optional) <i><targetGroupName></i> Use this argument to specify the new name to which the settings need to be moved. Argument type: String
<code>newName</code>	(Optional) Enter any name of your choice to rename the group. Argument type: String
<code>removeAllUsers</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If the value is <code>true</code> or <code>1</code> , the system removes all the users from this group. Argument type: Boolean
<code>userNames</code>	<code>user1 [user2...]</code> 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

Arguments	Descriptions
userName	<p>(Optional) Using ec-perl, enter one or more user names to add to the group.</p> <p>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.</p> <p>Argument type: Collection</p>
userName	<p>(Optional) Using ectool, enter one user name to add to the group.</p> <p>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.</p> <p>Argument type: String</p>
userNames	<p>(Optional) Using ectool, enter two or more user names to add to the group.</p> <p>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.</p> <p>Argument type: Collection</p>

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
<code>userName</code>	The name used by the user to log in and/or receive email. For example, "jsmith". Argument type: String
<code>email</code>	(Optional) The user's email address. Argument type: String
<code>fullUserName</code>	(Optional) The user's full name. For example, "John Smith". Argument type: String
<code>groupNames</code>	(Optional) <i>group1 [group2 ...]</i> Assigns the user to one or more groups and removes the user from any groups not included in the list. Argument type: Collection
<code>migrateSettings</code>	<i><targetUserName></i> The new name to which the settings need to be moved. Argument type: String

Arguments	Descriptions
<code>newName</code>	(Optional)The new name of the user (for example, changing an existing user's surname). Argument type: String
<code>password</code>	(Optional) Enter a new password for the user. Argument type: String
<code>removeFromAllGroups</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If set to 1 or true, this user will be removed from all groups. Argument type: String
<code>sessionPassword</code>	(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](#) on page 652

[createTransitionDefinition](#) on page 654

[createWorkflowDefinition](#) on page 655

[deleteStateDefinition](#) on page 656

[deleteTransitionDefinition](#) on page 657

[deleteWorkflowDefinition](#) on page 658

[getStateDefinition](#) on page 659

[getStateDefinitions](#) on page 660

[getTransitionDefinition](#) on page 661

[getTransitionDefinitions](#) on page 662

[getWorkflowDefinition](#) on page 663

[getWorkflowDefinitions](#) on page 664

[modifyStateDefinition](#) on page 664

[modifyTransitionDefinition](#) on page 666

[modifyWorkflowDefinition](#) on page 668

[moveStateDefinition](#) on page 669

[moveTransitionDefinition](#) on page 670

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
<code>projectName</code>	The name of the project. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>stateDefinitionName</code>	Choose any unique name of your choice for the state definition. This name must be unique within the workflow definition. Argument type: String
<code>actualParameters</code>	(Optional) Specifies the values to pass as parameters to the process. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the process. For more information about parameters, click here . Argument type: Map
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String

Arguments	Descriptions
startable	(Optional) <i><Boolean flag - 0 1 true false></i> – 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
<code>projectName</code>	The name for the project that must be unique among all projects. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>stateDefinitionName</code>	The name of the state definition. Argument type: String
<code>transitionDefinitionName</code>	The name of the transition that must be unique among all state definitions. Argument type: String
<code>targetState</code>	Target state for the transition definition. Argument type: String
<code>actualParameters</code>	(Optional) Specifies the values to pass as parameters to the target state. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the target state. For more information about parameters, click here . Argument type: Map
<code>condition</code>	(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 <code>trigger</code> is set to manual. Argument type: String

Arguments	Descriptions
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
trigger	<p>(Optional) Type of trigger for this transaction.</p> <p>Possible values are: <code>onEnter</code> <code>onStart</code> <code>onCompletion</code> <code>manual</code></p> <p>Argument type: TransitionTrigger</p>

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" --trigger 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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> 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
<code>projectName</code>	The name of the project containing the state definition. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>stateDefinitionName</code>	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
<code>projectName</code>	The name of the project containing the transition definition. Argument type: String

Arguments	Descriptions
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

Arguments	Descriptions
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
<code>projectName</code>	The name for the project that must be unique among all projects. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>includeFormalParameters</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If this argument is set to 1 or true, formal parameters will also be included in the response. Argument type: Boolean
<code>startableOnly</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument type: String
<code>workflowDefinitionName</code>	Name of the workflow definition. Argument type: String
<code>stateDefinitionName</code>	Name of the state definition. Argument type: String
<code>transitionDefinitionName</code>	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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument type: String
<code>workflowDefinitionName</code>	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
<code>projectName</code>	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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String

Arguments	Descriptions
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 <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called process. Argument type: Map
clearActualParameters	(Optional) <i><Boolean flag - 0 1 true false></i> If set to <code>true</code> or <code>1</code> , 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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String
newName	(Optional) New name for the state definition. Argument type: String
startable	(Optional) <i><Boolean flag - 0 1 true false></i> If this argument is set to <code>1</code> or <code>true</code> , 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 <code>subproject</code> . 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
substartingState	(Optional) The name of the workflow starting state that is launched when the state is entered. Using this argument also requires <code>subproject</code> and <code>subworkflowDefinition</code> . Argument type: String

Arguments	Descriptions
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 factory  
--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

Arguments	Descriptions
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>stateDefinitionName</code>	The name of the state definition. Argument type: String
<code>transitionDefinitionName</code>	The name of the transition definition to modify. Argument type: String
<code>actualParameter</code>	(Optional) Specifies the values to pass as parameters to the target state. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the target state.
<code>clearActualParameters</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If set to <code>true</code> or <code>1</code> , all the actual parameters are removed from the definition. Argument type: Boolean
<code>condition</code>	(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 <code>trigger</code> is set to manual. Argument type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String
<code>newName</code>	(Optional) New name of the transition definition that must be a unique name within the workflow. Argument type: String
<code>targetState</code>	(Optional) The target state for the transition definition. Argument type: String
<code>trigger</code>	(Optional) Possible values are: <code>onEnter onStart onCompletion manual</code> . 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
workflowDefinitionName	The name of the workflow definition to modify. Argument type: String

Arguments	Descriptions
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
newName	<p>(Optional) The new name for the workflow definition. It must be a unique name within the workflow.</p> <p>Argument type: String</p>
workflowNameTemplate	<p>(Optional) The template used to determine default names for workflows launched from a workflow definition.</p> <p>Argument type: String</p>

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
<code>projectName</code>	The name of the project containing the state definition. Argument type: String
<code>stateDefinitionName</code>	The name of the state definition to move. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>beforeStateDefinition</code>	(Optional) Use this option to reorder state definitions in a workflow definition. The state definition (<code>stateDefinitionName</code>) 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 "Build"
```

[Back to Top](#)

moveTransitionDefinition

Moves a transition definition within a workflow definition.

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

Arguments	Descriptions
<code>projectName</code>	The name of the project containing the transition definition. Argument type: String
<code>workflowDefinitionName</code>	The name of the workflow definition. Argument type: String
<code>stateDefinitionName</code>	The name of the state definition. Argument type: String
<code>transitionDefinitionName</code>	The name of the transition definition to move. Argument type: String
<code>beforeTransitionDefinition</code>	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](#) on page 672

[deleteWorkflow](#) on page 673

[getState](#) on page 674

[getStates](#) on page 674

[getTransition](#) on page 675

[getTransitions](#) on page 676

[getWorkflow](#) on page 677

[getWorkflows](#) on page 678

[runWorkflow](#) on page 679

[transitionWorkflow](#) on page 680

completeWorkflow

Marks a workflow as completed. When completed, transitions are no longer evaluated.

You must specify `projectName` and `workflowName`.

Arguments	Descriptions
<code>projectName</code>	The name for the project that must be unique among all projects. Argument type: String
<code>workflowName</code>	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
<code>projectName</code>	The name of the project containing the workflow to delete. Argument type: String
<code>workflowName</code>	The name of the workflow. Argument type: String
<code>deleteProcesses</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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 => true});
```

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
<code>projectName</code>	Name for the project; must be unique among all projects. Argument type: String
<code>workflowName</code>	Name of the workflow. Argument type: String
<code>stateName</code>	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", "build2test");
```

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
<code>projectName</code>	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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument type: String
<code>workflowDefinitionName</code>	Name of the workflow definition. Argument type: String
<code>actualParameter</code>	(Optional) Specifies the values to pass as parameters to the workflow starting state. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the starting state. Argument type: Map
<code>credentials</code>	(Optional) Credentials to use with the workflow state. Argument type: Collection
<code>priority</code>	(Optional) Priority of the jobs launched by the workflow. Argument type: JobPriority
<code>startingState</code>	(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
<code>projectName</code>	Name for the project that must be unique among all projects. Argument type: String
<code>workflowName</code>	The name of the workflow to transition. Argument type: String
<code>stateName</code>	The name of the new state. Argument type: String
<code>transitionName</code>	The name of the transition. Argument type: String
<code>actualParameters</code>	(Optional) Specifies the values to pass as parameters to the transition's target state. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the target state. Argument type: Map
<code>credentials</code>	(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](#) on page 681

[deleteWorkspace](#) on page 683

[getWorkspace](#) on page 684

[getWorkspaces](#) on page 685

[modifyWorkspace](#) on page 685

[resolveFile](#) on page 688

createWorkspace

Creates a new workspace.

A workspace definition consists of three paths to access the workspace in various ways:

agentDrivePath

agentUncPath - The agent uses agentUncPath and agentDrivePath to compute the drive mapping needed to make agentDrivePath valid in the step (see examples below).

agentUnixPath

Examples for agentDrivePath and agentUncPath:

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

You must specify a `workspaceName`.

Arguments	Descriptions
<code>workspaceName</code>	The workspace name. Argument type: String
<code>agentDrivePath</code>	(Optional) Drive-letter-based path used by Windows agents to access the workspace in steps. Argument type: String
<code>agentUncPath</code>	(Optional) UNC path used by Windows ElectricFlow web servers to access the workspace. The agent uses <code>agentUncPath</code> and <code>agentDrivePath</code> to compute the drive mapping needed for making <code>agentDrivePath</code> valid in the step. Argument type: String
<code>agentUnixPath</code>	(Optional) UNIX path used by UNIX agents and Linux ElectricFlow web servers to access the workspace. Argument type: String
<code>credentialName</code>	(Optional) Credential to use when connecting to a network location. <code>credentialName</code> can be one of two forms: relative (for example, " <code>cred1</code> ") - the credential is assumed to be in the project that contains the request target object absolute (for example, " <code>/projects/BuildProject/credentials/cred1</code> ") - the credential can be from any specified project, regardless of the target object's project. Argument type: String
<code>description</code>	(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code> Argument type: String
<code>local</code>	(Optional) <i><Boolean flag - 0 1 true false></i> When the value is 1 or true, the workspace is local. Argument type: String

Arguments	Descriptions
<code>workspaceDisabled</code>	(Optional) <Boolean flag - 0 1 true false> When the value is 1 or true, the workspace is disabled. Argument type: String
<code>zoneName</code>	(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
<code>workspaceName</code>	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
<code>workspaceName</code>	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 <code>\\server\share</code> to drive <code>n:</code> and runs the step in <code>n:\job123</code> .

agentDrivePath	agentUncPath	Result from running a step in "job123" that uses this workspace
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
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

Arguments	Descriptions
description	<p>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code></p> <p>Argument type: String</p>
local	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If the value is 1 or true, the workspace is local.</p> <p>Argument type: Boolean</p>
newName	<p>(Optional) Enter any name of your choice to rename the workspace.</p> <p>Argument type: String</p>
workspaceDisabled	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If the value is 1 or true, the workspace is disabled.</p> <p>Argument type: Boolean</p>
zoneName	<p>(Optional) The name of the zone where this workspace resides.</p> <p>Argument type: String</p>

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
<code>fromAgentId</code>	Identifier of the agent requesting the route to a destination agent or artifact repository. Argument type: Long
<code>workspaceName</code>	Name of the workspace. Argument type: String
<code>jobId</code>	(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
<code>jobStepId</code>	(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
<code>resourceName</code>	(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](#) on page 689[changeOwner](#) on page 690[clone](#) on page 693[countObjects](#) on page 699[deleteObjects](#) on page 702[dumpHeap](#) on page 706[dumpStatistics](#) on page 706[evalDsl](#) on page 708[evalScript](#) on page 713[export](#) on page 718[findObjects](#) on page 722[finishCommand](#) on page 728[generateDsl](#) on page 730[getObjects](#) on page 730[graphStateMachine](#) on page 732[import](#) on page 733[logStatistic](#) on page 736[releaseNamedLock](#) on page 737

acquireNamedLock

Retrieves the named lock.

Arguments	Descriptions
<code>lockName</code>	Name of the lock. Argument type: String
<code>create</code>	<i><Boolean flag - 0 1 true false></i> When this argument is set to <code>true</code> or <code>1</code> , 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
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

Arguments	Descriptions
<code>credentialName</code>	<p>(Optional) <code>credentialName</code> can be in one of these formats:</p> <ul style="list-style-type: none"> • relative (for example, "<code>cred1</code>") - the credential is assumed to be in the project that contains the request target object. Requires a qualifying project name. • absolute (for example, "<code>/projects/BuildProject/credentials/cred1</code>") - the credential can be from any specified project, regardless of the target object's project. <p>Argument type: String</p>
<code>environmentName</code>	<p>(Optional) The name of an environment.</p> <p>Argument type: String</p>
<code>environmentTemplateName</code>	<p>(Optional) The name of an environment template.</p> <p>Argument type: String</p>
<code>environmentTierName</code>	<p>(Optional) The name of an environment tier.</p> <p>Argument type: String</p>
<code>groupName</code>	<p>(Optional) The full name of a group. For Active Directory and LDAP, this is a full domain name.</p> <p>Argument type: String</p>
<code>newOwnerName</code>	<p>(Optional) The name of the new owner for this object. This defaults to the current user.</p> <p>Argument type: String</p>
<code>notifierName</code>	<p>(Optional) The name of the email notifier.</p> <p>Argument type: String</p>
<code>pluginName</code>	<p>(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.</p> <p>Argument type: String</p>
<code>procedureName</code>	<p>(Optional) The name of the procedure. It can be a path to the procedure. When using this argument, you must also enter the <code>projectName</code>.</p> <p>Argument type: String</p>
<code>processName</code>	<p>(Optional) The name of a process. It can be a path to the process.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>processStepName</code>	(Optional) The name of a process step. It can be a path to the process step. Argument type: String
<code>projectName</code>	(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
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID
<code>resourceName</code>	(Optional) The name of the resource. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule. It can be a path to the schedule. When using this argument, you must also use <code>projectName</code> . Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stepName</code>	(Optional) The name of the step. It can be a path to the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>transitionDefinitionName</code>	(Optional) The name of the transition definition. Argument type: String
<code>userName</code>	(Optional) The full name of the user. For Active Directory and LDAP, the name can be <code>user@domain</code> . Argument type: String
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workspaceName</code>	(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	
<code>cloneName</code>	<p>(Optional) New name of the cloned copy of an object. The <code>cloneName</code> specifies the path to the new object, possibly in an alternate location.</p> <p>If no container path is specified, the new object is created inside the same container as the original.</p> <p>If no name is specified, the server will generate a name.</p> <p>Argument type: String</p>
Optionals	
<code>disableProjectTracking</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If set to <code>true</code>, 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.</p> <p>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.</p> <p>Argument type: Boolean</p>

Arguments	Descriptions
<code>reducedDetailChangeHistory</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>Use this argument for large projects containing over 20,000 audited objects with Change Tracking enabled.</p> <p>When this argument is set to <code>true</code> or <code>1</code>, 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.</p> <p>Set this argument to <code>false</code> or <code>0</code> 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.</p> <p>Argument type: Boolean</p>
Locators	
<code>applicationName</code>	<p>(Optional) The name of the application that is unique among all projects.</p> <p>Argument type: String</p>
<code>componentName</code>	<p>(Optional) The name of the component.</p> <p>Argument type: String</p>
<code>configName</code>	<p>(Optional) The name of the email configuration (<code>emailConfig</code>).</p> <p>Argument type: String</p>
<code>credentialName</code>	<p>(Optional) The name of the credential that can be specified in one of these formats:</p> <p>relative (for example, "<code>cred1</code>")—The credential is assumed to be in the project that contains the target object.</p> <p>absolute(for example, "<code>/projects/BuildProject/credentials/cred1</code>")—The credential can be from any specified project, regardless of the project with the target object.</p> <p>Argument type: String</p>
<code>environmentTemplateName</code>	<p>(Optional) The name of the environment template.</p> <p>Argument type: String</p>
<code>flowName</code>	<p>(Optional) Name of the flow that must be unique within the project.</p> <p>Argument Type: String</p>

Arguments	Descriptions
<code>flowRuntimeName</code>	(Optional) Name of the flow runtime. Argument Type: String
<code>flowRuntimeStateName</code>	(Optional) Name of the flow state. Argument Type: String
<code>flowStateName</code>	(Optional) Name of the flow state that must be unique within the flow. Argument Type: String
<code>flowTransitionName</code>	Name of the flow transition that must be unique within the flow state. Argument Type: String
<code>notifierName</code>	(Optional) The name of the email notifier (<code>emailNotifier</code>). Argument type: String
<code>objectId</code>	(Optional) The object id as returned by <code>findObjects</code> . Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(Optional) The name of the plugin. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure that you want to clone. When using this argument, you must also enter the <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process. Argument type: String
<code>projectName</code>	<p>(Optional) The name of the project that you want to clone.</p> <div> <p>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.</p> </div> <p>Argument type: String</p>

Arguments	Descriptions
<code>propertySheetId</code>	(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created.
<code>providerName</code>	(Optional) The unique name of the directory provider, such as the LDAP or Active Directory provider name. Argument type: String
<code>releaseName</code>	(Optional) The name of the Release which owns the property. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource that you want to clone. Argument type: String
<code>resourcePoolName</code>	(Optional) The name of a resource pool. Argument type: String
<code>resourceTemplateName</code>	(Optional) The name of the resource template. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule that you want to clone. When using this argument, you must also enter <code>projectName</code> . Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stepName</code>	(Optional) The name of the procedure step that you want to clone. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>transitionDefinitionName</code>	(Optional) The name of the transition definition. Argument type: String
<code>workflowDefinitionName</code>	(Optional) The name of the workflow definition. Argument type: String
<code>workspaceName</code>	(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/setProperty'
<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	<p>The type of object to count.</p> <p>Values include:</p> <table> <tr><td>application</td><td>process</td></tr> <tr><td>applicationTier</td><td>processDependency</td></tr> <tr><td>artifact</td><td>processStep</td></tr> <tr><td>artifactVersion</td><td>project</td></tr> <tr><td>component</td><td>property</td></tr> <tr><td>credential</td><td>release</td></tr> <tr><td>deployerApplication</td><td>repository</td></tr> <tr><td>deployerConfiguration</td><td>resource</td></tr> <tr><td>directoryProvider</td><td>resourcePool</td></tr> <tr><td>emailConfig</td><td>resourceTemplate</td></tr> <tr><td>emailNotifier</td><td>schedule</td></tr> <tr><td>environment</td><td>snapshot</td></tr> <tr><td>environmentTemplate</td><td>stage</td></tr> <tr><td>environmentTemplateTier</td><td>state</td></tr> <tr><td>environmentTier</td><td>stateDefinition</td></tr> <tr><td>formalParameter</td><td>task</td></tr> <tr><td>job</td><td>tierMap</td></tr> <tr><td>jobStep</td><td>transition</td></tr> <tr><td>logEntry</td><td>transitionDefinition</td></tr> <tr><td>pipeline</td><td>workflow</td></tr> <tr><td>plugin</td><td>workflowDefinition</td></tr> <tr><td>procedure</td><td>workspace</td></tr> <tr><td>procedureStep</td><td></td></tr> </table> <p>Argument type: String</p>	application	process	applicationTier	processDependency	artifact	processStep	artifactVersion	project	component	property	credential	release	deployerApplication	repository	deployerConfiguration	resource	directoryProvider	resourcePool	emailConfig	resourceTemplate	emailNotifier	schedule	environment	snapshot	environmentTemplate	stage	environmentTemplateTier	state	environmentTier	stateDefinition	formalParameter	task	job	tierMap	jobStep	transition	logEntry	transitionDefinition	pipeline	workflow	plugin	workflowDefinition	procedure	workspace	procedureStep	
application	process																																														
applicationTier	processDependency																																														
artifact	processStep																																														
artifactVersion	project																																														
component	property																																														
credential	release																																														
deployerApplication	repository																																														
deployerConfiguration	resource																																														
directoryProvider	resourcePool																																														
emailConfig	resourceTemplate																																														
emailNotifier	schedule																																														
environment	snapshot																																														
environmentTemplate	stage																																														
environmentTemplateTier	state																																														
environmentTier	stateDefinition																																														
formalParameter	task																																														
job	tierMap																																														
jobStep	transition																																														
logEntry	transitionDefinition																																														
pipeline	workflow																																														
plugin	workflowDefinition																																														
procedure	workspace																																														
procedureStep																																															

Arguments	Descriptions
filter	<p>(Optional) A list of zero or more filter criteria definitions used to define objects to find.</p> <p>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.</p> <p>Two types of filters:</p> <p>"property filters" - used to select objects based on the value of the object's intrinsic or custom property</p> <p>"boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic.</p> <p>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.</p> <p>Property filter operators are:</p> <pre> 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) </pre> <p>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.</p> <p>Boolean operators are:</p> <pre> not (1 operand) and (2 or more operands) or (2 or more operands) </pre> <p>Argument type: Collection</p>

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
            "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
<code>objectType</code>	<p>This argument specifies the type of object to find.</p> <p>Values include:</p> <p><code>artifact artifactVersion job logEntry project repository workflow</code></p> <p>Argument type: String</p>

Arguments	Descriptions
filters	<p>(Optional) Specify filters in a space-separated list: <code>filter1 filter2 ...</code> A list of zero or more filter criteria definitions used to define objects to find.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>Property filter operators are:</p> <pre> 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) </pre> <p>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.</p> <p>Boolean operators are:</p> <pre> not (1 operand) and (2 or more operands) or (2 or more operands) </pre> <p>Argument type: Collection</p>

Arguments	Descriptions
<code>maxIds</code>	<p>(Optional) <code><id count></code> The maximum number of objects that will be deleted. The default is all objects that match the filter.</p> <p>Argument type: Integer</p>
<code>sorts</code>	<p>(Optional) Specify "sorts" in a space-separated list: <code>sort1 sort2 ...</code></p> <p>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.</p> <p>If you specify more than one sort criterion, the sorts are applied according to the order they appear in the list.</p> <p>The first item in the list is the primary sort key.</p> <p>Each item in the list is a hash reference.</p> <p>See the code example below for instructions about forming the list and passing it to the ElectricFlow Perl API.</p> <p>The sort order affects which objects are deleted if a <code>maxIds</code> value limits the number of objects returned by the filter.</p> <p>Argument type: Collection</p>

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
<code>clearStatistics</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to 1 or true, the system clears the statistics after logging them.</p> <p>Argument type: Boolean</p>
<code>dumpLapTimes</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to 1 or true, the system dumps lap times.</p> <p>If this argument is set to 0 or false, the system dumps global times.</p> <p>Argument type: Boolean</p>
<code>fileName</code>	<p>(Optional) If you specify a file name with a path, the output is redirected to this file.</p> <p>When the path is relative, the file is written relative to the working directory of the server.</p> <p>Argument type: String</p>
<code>format</code>	<p>(Optional) Format of the output.</p> <p>Valid values are <code>text</code> or <code>xml</code>.</p> <p>The default is <code>.</code></p> <p>Argument type: String <code>text</code></p>

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 the `dsl` argument.

Arguments	Descriptions
<code>dsl</code>	The DSL text. Argument type: String
<code>applicationName</code>	(Optional) Name of the application. Argument type: String
<code>applicationTierName</code>	(Optional) Name of the application tier. Argument type: String
<code>artifactName</code>	(Optional) Name of the artifact. Argument type: String
<code>artifactVersionName</code>	(Optional) Name of the artifact version. Argument type: String
<code>componentName</code>	(Optional) Name of the component. Argument type: String
<code>configName</code>	(Optional) Name of the email configuration. Argument type: String
<code>credentialName</code>	(Optional) Name of the credential. Argument type: String
<code>debug</code>	(Optional) <Boolean flag - 0 1 true false> If this argument is set to <code>true</code> or <code>1</code> , ElectricFlow generates debug output as the DSL script is evaluated. Argument type: Boolean
<code>describe</code>	(Optional) <Boolean flag - 0 1 true false> If this argument is set to <code>true</code> or <code>1</code> , ElectricFlow prints a description of the DSL text. Argument type: Boolean

Arguments	Descriptions
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
jobStepId	(Optional) The primary key or name of the job step. Argument type: UUID
notifierName	(Optional) The name of the notifier. Argument type: UUID

Arguments	Descriptions
<code>objectId</code>	(Optional) The object ID returned by the <code>findObjects</code> call. Argument type: UUID
<code>parameters</code>	(Optional) Parameters are passed to the script by ElectricFlow as JSON text. Argument type: String
<code>path</code>	(Optional) The property path. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(Optional) The name of the plugin. Argument type: String
<code>procedureName</code>	(Optional) The name of the procedure. Argument type: String
<code>processName</code>	(Optional) The name of the process if the container is a process or process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step if the container is a process step. Argument type: String
<code>projectName</code>	(Optional) The name of the project. Argument type: String
<code>propertySheetId</code>	(Optional) The primary key or name of the property sheet. Argument type: UUID
<code>releaseName</code>	(Optional) The name of the Release. Argument type: String
<code>repositoryName</code>	(Optional) The name of the repository. Argument type: String
<code>resourceName</code>	(Optional) The name of the resource. Argument type: String

Arguments	Descriptions
<code>resourcePoolName</code>	(Optional) The name of the resource pool. Argument type: String
<code>resourceTemplateName</code>	(Optional) The name of the resource template. Argument type: String
<code>scheduleName</code>	(Optional) The name of the schedule. Argument type: String
<code>serverLibraryPath</code>	(Optional) Path to the server directory that contains <code>.jar</code> files and classes to be added to the classpath when evaluating the DSL text. Argument type: String
<code>sessionPassword</code>	(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
<code>snapshotName</code>	(Optional) The name of the snapshot that you want to clone. Argument type: String
<code>stageName</code>	(Optional) The name of the stage definition. Argument type: String
<code>stateDefinitionName</code>	(Optional) The name of the state definition. Argument type: String
<code>stateName</code>	(Optional) The name of the state. Argument type: String
<code>stepName</code>	(Optional) The name of the step that you want to clone. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code> . Argument type: String
<code>systemObjectName</code>	(Optional) System object names include: <code>admin artifacts directory emailConfigs forceAbort licensing log plugins priority projects repositories resources server session workspaces zonesAndGateways</code> 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 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>, {<optionals>});

Example

```
$cmdr->evalDsl ("Run the sample application", {dslFile => run_deploy_application.groovy});
```

ectool

syntax: ectool evalDsl <dsl> [optionals]

Example

```
ectool evalDsl "Run the sample application" --dslFile run_deploy_application.groovy
```

[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
<code>value</code>	The script to evaluate in the specified context. Argument type: String
<code>applicationName</code>	(Optional) The name of the application that must be unique among all projects. Argument type: String
<code>applicationTierName</code>	(Optional) The name of the application tier. Argument type: String
<code>artifactName</code>	(Optional) The name of the artifact. Argument type: String
<code>artifactVersionName</code>	(Optional) The name of the artifact version. Note: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as <code>"groupId:artifactKey:version"</code> and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly. Argument type: String
<code>componentName</code>	(Optional) Name of the component. Argument type: String
<code>configName</code>	(Optional) Name of the email configuration. Argument type: String

Arguments	Descriptions
credentialName	<p>(Optional) Name of the credential in one of these forms:</p> <ul style="list-style-type: none"> • relative (for example, "cred1")—The credential is assumed to be in the project that contains the request target object. • absolute (for example, "/projects/BuildProject/credentials/cred1")—The credential can be from any specified project, regardless of the target object's project. <p>Argument type: String</p>
environmentName	<p>(Optional) The name of the environment that must be unique among all projects.</p> <p>Argument type: String</p>
environmentTemplateName	<p>(Optional) The name of the environment template.</p> <p>Argument type: String</p>
environmentTemplateTierName	<p>(Optional) The name of the environment template tier.</p> <p>Argument type: String</p>
environmentTierName	<p>(Optional) The name of the environment tier.</p> <p>Argument type: String</p>
flowName	<p>(Optional) The name of the flow.</p> <p>Argument type: String</p>
flowRuntimeName	<p>(Optional) Name of the flow runtime.</p> <p>Argument Type: String</p>
flowRuntimeStateName	<p>(Optional) Name of the flow state.</p> <p>Argument Type: String</p>
flowStateName	<p>(Optional) The name of the flow state.</p> <p>Argument type: String</p>
flowTransitionName	<p>(Optional) The name of the flow transition.</p> <p>Argument type: String</p>
gatewayName	<p>(Optional) The name of the gateway.</p> <p>Argument type: String</p>
groupName	<p>(Optional) The name of a group where you might evaluate a script.</p> <p>Argument type: String</p>

Arguments	Descriptions
<code>jobId</code>	(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
<code>jobStepId</code>	(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID
<code>notifierName</code>	(Optional) The name of the email notifier. Argument type: String
<code>objectId</code>	(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code> . Argument type: String
<code>path</code>	(Optional) The property path. Argument type: String
<code>pipelineName</code>	(Optional) The name of the pipeline. Argument type: String
<code>pluginName</code>	(Optional) The name of a plugin where you might evaluate a script. Argument type: String
<code>procedureName</code>	(Optional) The name of a procedure where you might need to evaluate a script. Also requires <code>projectName</code> . Argument type: String
<code>processName</code>	(Optional) The name of the process when the container is a process or process step. Argument type: String
<code>processStepName</code>	(Optional) The name of the process step when the container is a process step. Argument type: String
<code>projectName</code>	(Optional) Name of the project that contains the script to evaluate. 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. 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
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

Arguments	Descriptions
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-73f1-11e3-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-b0a420524153
--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.

IMPORTANT: The export operation is run by the server process, not through ectool. 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.

Arguments	Descriptions
<code>fileName</code>	<p><code><remoteFileName></code> The specified directory for the file must already exist in the system. If the path is local, it will be created on the server. If it is a network path, it must be accessible by the server and the server user.</p> <p>Argument type: String</p>

Arguments	Descriptions
compress	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>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.</p> <p>Note: This is true only for full exports.</p> <p>Argument type: Boolean</p>
disableProjectTracking	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>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.</p> <p>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.</p> <p>Note: This is true only for partial exports.</p> <p>Argument type: Boolean</p>
download	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>If set to 1 or true, the exported file can be downloaded.</p> <p>Note: This is true for full and partial exports.</p> <p>Argument type: Boolean</p>
excludeJobs	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>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.</p> <p>Note: This is true only for full exports.</p> <p>Argument type: Boolean</p>
objectId	<p>(Optional) ID of the object ID.</p> <p>Note: This triggers a partial export.</p> <p>Argument type: UUID</p>
objectType	<p>(Optional) The object type.</p> <p>Note: This triggers a partial export.</p> <p>Argument type: String</p>

Arguments	Descriptions
path	<p>(Optional) <i><property path></i> 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.</p> <p>Note: This triggers a partial export.</p> <p>Argument type: String</p>
reducedDetailChangeHistory	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>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.</p> <p>Argument type: Boolean</p>
relocatable	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If the <code>--relocatable</code> flag is set to 1 or true, a partial export (for example, with <code>--path</code>) will not include object IDs, ACLs, system property sheets, create/modify times, owners, email notifiers or <code>lastModifiedBy</code> 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.</p> <p>Note: The relocatable argument only works with a partial export. This argument is silently ignored during a full export.</p> <p>Argument type: Boolean</p>
revisionNumber	<p>(Optional) Revision number of the file.</p> <p>Note: This triggers a partial export.</p> <p>Argument type: Integer</p>
safeMode	<p>(Optional) The <code>safeMode</code> argument determines whether the server will be quiesced before a full export begins and if yes, whether or not the server will shutdown and restarted after the export completes. Values are:</p> <ul style="list-style-type: none"> • 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. <p>Note: The <code>safeMode</code> argument affects only full imports.</p> <p>Argument type: SafeMode</p>

Arguments	Descriptions
<code>withAcls</code>	<p>(Optional) This argument modifies <code>relocatable</code>. <i><Boolean flag - 0 1 true false></i></p> <p>If the <code>withAcls</code> flag is set to 1 or <code>true</code>, a relocatable partial export will include ACLs.</p> <p>Note: This is true only for partial exports.</p> <p>Argument type: Boolean</p>
<code>withNotifiers</code>	<p>(Optional) This argument modifies <code>relocatable</code>. <i><Boolean flag - 0 1 true false></i></p> <p>If this flag is set to 1 or <code>true</code>, a relocatable partial export will include email notifiers.</p> <p>Note: This is true only for partial exports.</p> <p>Argument type: Boolean</p>
<code>withVersionNumbers</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this flag is set to 1 or <code>true</code>, version numbers are issued during the export operation (they are ignored on import).</p> <p>Note: This is true only for partial exports.</p> <p>Argument type: Boolean</p>

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
```

```
Projl"  
  --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																																
<code>objectType</code>	<p>The type of object to find. Values include:</p> <table><tr><td><code>application</code></td><td><code>process</code></td></tr><tr><td><code>artifact</code></td><td><code>processStep</code></td></tr><tr><td><code>artifactVersion</code></td><td><code>project</code></td></tr><tr><td><code>component</code></td><td><code>property</code></td></tr><tr><td><code>credential</code></td><td><code>repository</code></td></tr><tr><td><code>directoryProvider</code></td><td><code>resource</code></td></tr><tr><td><code>emailconfig</code></td><td><code>resourcePool</code></td></tr><tr><td><code>emailNotifier</code></td><td><code>schedule</code></td></tr><tr><td><code>environment</code></td><td><code>state</code></td></tr><tr><td><code>formalParameter</code></td><td><code>stateDefinition</code></td></tr><tr><td><code>job</code></td><td><code>transition</code></td></tr><tr><td><code>jobStep</code></td><td><code>transitionDefinition</code></td></tr><tr><td><code>logEntry</code></td><td><code>workflow</code></td></tr><tr><td><code>plugin</code></td><td><code>workflowDefinition</code></td></tr><tr><td><code>procedure</code></td><td><code>workspace</code></td></tr><tr><td><code>procedureStep</code></td><td></td></tr></table> <p>Argument type: String</p>	<code>application</code>	<code>process</code>	<code>artifact</code>	<code>processStep</code>	<code>artifactVersion</code>	<code>project</code>	<code>component</code>	<code>property</code>	<code>credential</code>	<code>repository</code>	<code>directoryProvider</code>	<code>resource</code>	<code>emailconfig</code>	<code>resourcePool</code>	<code>emailNotifier</code>	<code>schedule</code>	<code>environment</code>	<code>state</code>	<code>formalParameter</code>	<code>stateDefinition</code>	<code>job</code>	<code>transition</code>	<code>jobStep</code>	<code>transitionDefinition</code>	<code>logEntry</code>	<code>workflow</code>	<code>plugin</code>	<code>workflowDefinition</code>	<code>procedure</code>	<code>workspace</code>	<code>procedureStep</code>	
<code>application</code>	<code>process</code>																																
<code>artifact</code>	<code>processStep</code>																																
<code>artifactVersion</code>	<code>project</code>																																
<code>component</code>	<code>property</code>																																
<code>credential</code>	<code>repository</code>																																
<code>directoryProvider</code>	<code>resource</code>																																
<code>emailconfig</code>	<code>resourcePool</code>																																
<code>emailNotifier</code>	<code>schedule</code>																																
<code>environment</code>	<code>state</code>																																
<code>formalParameter</code>	<code>stateDefinition</code>																																
<code>job</code>	<code>transition</code>																																
<code>jobStep</code>	<code>transitionDefinition</code>																																
<code>logEntry</code>	<code>workflow</code>																																
<code>plugin</code>	<code>workflowDefinition</code>																																
<code>procedure</code>	<code>workspace</code>																																
<code>procedureStep</code>																																	

Arguments	Descriptions
filters	<p>(Optional) A list of zero or more filter criteria definitions used to define objects to find.</p> <p>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.</p> <p>Two types of filters:</p> <p>"property filters" - used to select objects based on the value of the object's intrinsic or custom property</p> <p>"boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic.</p> <p>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.</p> <p>Property filter operators are:</p> <pre> 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) </pre> <p>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.</p> <p>Boolean operators are:</p> <pre> not (1 operand) and (2 or more operands) or (2 or more operands) </pre> <p>Argument type: Collection</p>
firstResult	<p>(Optional) The first result to be retrieved, starting from 0 (zero).</p> <p>Argument type: Integer</p>

Arguments	Descriptions
<code>includeAccess</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If set to 1 or true, this command also returns access maps for the objects.</p> <p>Argument type: Boolean</p>
<code>includeEntityRevisions</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If set to 1 or true, this command also returns the version and/or entity revision if the object is in the search results.</p> <p>Argument type: Boolean</p>
<code>includeLatestRevision</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If set to 1 or true, this command also returns the latest revision data for versioned objects.</p> <p>Argument type: Boolean</p>
<code>maxIds</code>	<p>(Optional) <i><id count></i></p> <p>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.</p> <p>Argument type: Integer</p>
<code>numObjects</code>	<p>(Optional) <i><full object count></i></p> <p>Specifies the number of full objects (not just the IDs) returned from the <code>findObjects</code> 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 <code>numObjects</code> is not specified, all full objects in the list of object IDs are returned. Any and all objects can be retrieved using the <code>getObjects</code> command.</p> <p>Argument type: Integer</p>
<code>selects</code>	<p>(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.</p> <p>Argument type: Collection</p>

Arguments	Descriptions
sorts	<p>(Optional) This is an ordered list of sort criteria. This option works only when you specify a property name.</p> <p>Each list entry consists of a property name and a sort order—either an ascending or descending sort order.</p> <p>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.</p> <p>See the code example below for instructions on forming the list and passing it to the ElectricFlow Perl API.</p> <p>Argument type: Collection</p>

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
<code>path</code>	A property path indicating a single object for which DSL will be generated Argument type: String
<code>withAcls</code>	(Optional) <i><Boolean flag - 0 1 true false></i> If set to <code>true</code> , 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 797.

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
<code>includeAccess</code>	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>If set to 1 or true, this command also returns access maps for the objects.</p> <p>Argument type: Boolean</p>
<code>includeLatestRevision</code>	<p>(Optional) <Boolean flag - 0 1 true false></p> <p>If set to 1 or true, this command also returns the latest revision data for versioned objects.</p> <p>Argument type: Boolean</p>
<code>objectIds</code>	<p>(Optional) A list of one or more object IDs that were returned by a prior call to <code>findObjects</code>. 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.</p> <p>Argument Type: Collection</p>
<code>selects</code>	<p>(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.</p> <p>Argument type: Collection</p>

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
<code>jobId</code>	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
<code>cluster</code>	(Optional) <i><Boolean flag - 0 1 true false></i> 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.

Arguments	Descriptions
fileName	<p>Name of the file to import:</p> <p><i><remoteFileName></i> 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.</p> <p><i><localFileName></i> This is the path to a file on the client to import. The file is uploaded from the client to the server. The specified <i><file></i> value is sent as an attachment to the <code>import</code> API request. The server detects the presence of the attachment and reads the attached file instead of looking for a file on the server. The maximum file size specified by <code>file</code> is determined by the maximum upload-size server setting.</p> <p>By default the limit is 50MB, so this option should be used only for individually exported objects, not a full system export.</p> <p>Argument type: String</p>
batchSize	<p>(Optional) <i><batch size></i> 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.</p> <p>Note: The <code>batchSize</code> argument applies only to full import operations.</p> <p>Argument type: Integer</p>
disableProjectTracking	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to 1 or <code>true</code> 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.</p> <p>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.</p> <p>Argument type: Boolean</p>
disableSchedules	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to 1 or <code>true</code>, imported schedules will be disabled. This argument can modify imported schedules after import and before they are used to start a job.</p> <p>Argument type: Boolean</p>

Arguments	Descriptions
<code>force</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to <code>1</code> or <code>true</code>, 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.</p> <p>Argument type: Boolean</p>
<code>path</code>	<p>(Optional) <i><property path></i></p> <p>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 <code>path</code> option.</p> <p>Argument type: String</p>
<code>preserveId</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>If this argument is set to <code>1</code> or <code>true</code>, this command tries to preserve the <code>objectId</code> during a single-object import. If you are doing a <i>partial</i> import, using this option preserves the original job ID or workflow ID.</p> <p>Argument type: Boolean</p>
<code>reducedDetailChangeHistory</code>	<p>(Optional) <i><Boolean flag - 0 1 true false></i></p> <p>Use this argument for large projects containing over 20,000 audited objects with Change Tracking enabled.</p> <p>When this argument is set to <code>true</code> or <code>1</code>, 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.</p> <p>Set this argument to <code>false</code> or <code>0</code> 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.</p> <p>Argument type: Boolean</p>

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/fullBackup.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 an * (asterik) 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:

<code>artifactKey</code>	<code>majorMinorPatch</code>
<code>artifactName</code>	<code>modifyTime</code>
<code>artifactVersionId</code>	<code>owner</code>
<code>artifactVersionName</code>	<code>propertySheetId</code>
<code>artifactVersionState</code>	<code>publisherJobId</code>
<code>buildNumber</code>	<code>publisherJobName</code>
<code>createTime</code>	<code>publisherJobStepId</code>

<code>dependentArtifactVersion</code>	<code>qualifier</code>
<code>description</code>	<code>repositoryName</code>
<code>groupId</code>	<code>retrievers</code>
<code>lastModifiedBy</code>	<code>version</code>

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 user name.

Contents:

findGroupsTest
 testResult
 details

```
    groupList
      group*
  findUserTest
    testResult
    details
    userList
      userInfo*
      email
      fullUserName
      mutable
      providerName
  userAuthenticationTest
    testResult
    details
```

emailConfig

Contains information about the configuration used to communicate with an email server.

Contents:

```
configName
createTime
description
emailConfigId
emailConfigName
lastModifiedBy
mailFrom
mailHost
mailPort
mailProtocol
mailUser
modifyTime
owner
propertySheetId
```

emailNotifier

Contains information about an email notifier.

Contents:

`condition`
`configName`
`container`
`createTime`
`description`
`destinations`
`emailNotifierId`
`eventType`
`formattingTemplate`
`lastModifiedBy`
`modifyTime`
`notifierName`
`owner`
`propertySheetId`

formalParameter

Contains information about a formal parameter.

Contents:

`container`
`createTime`
`defaultValue`
`description`
`expansionDeferred`
`formalParameterId`
`formalParameterName`
`lastModifiedBy`
`modifyTime`
`owner`
`required`
`type`

gateway

Contains information about a gateway.

Contents:

createTime
description
gatewayDisabled
gatewayId
gatewayName
hostName1
hostName2
lastModifiedBy
modifyTime
owner
port1
port2
propertySheetId
resourceName1
resourceName2

group

Contains information about a defined group of users.

Contents:

createTime
groupId
groupName
lastModifiedBy
modifyTime
mutable
owner
propertySheet
propertySheetId
providerName
users

job

Contains information about a running or completed job. Different API calls will result in different subsets of possible properties on the job. Refer to the specific API for details.

Contents:

<code>abortedBy</code>	<code>licenseWaitTime</code>
<code>abortStatus</code>	<code>liveProcedure</code>
<code>actualParameters*</code>	<code>liveSchedule</code>
<code>callingState</code>	<code>modifyTime</code>
<code>combinedStatus</code>	<code>outcome</code>
<code>createTime</code>	<code>owner</code>
<code>credentialName</code>	<code>priority</code>
<code>deleted</code>	<code>procedureName</code>
<code>directoryName</code>	<code>projectName</code>
<code>elapsedTime</code>	<code>propertySheet</code>
<code>errorCode</code>	<code>propertySheetId</code>
<code>errorMessage</code>	<code>resourceWaitTime</code>
<code>external</code>	<code>runAsUser</code>
<code>finish</code>	<code>scheduleName</code>
<code>jobId</code>	<code>start</code>
<code>jobName</code>	<code>status</code>
<code>jobStep*</code>	<code>steps</code>
<code>lastModifiedBy</code>	<code>totalWaitTime</code>
<code>launchedByUser</code>	<code>workspaceWaitTime</code>

jobStep

Contains information to define or locate a job step. Notice that the `calledProcedure` element (subcontainer element) can contain multiple `jobStep` elements.

Contents:

<code>abortedBy</code>	<code>outcome</code>
<code>abortStatus</code>	<code>owner</code>
<code>actualParameters</code>	<code>parallel</code>
<code>alwaysRun</code>	<code>postExitCode</code>
<code>assignedResourceName</code>	<code>postLogFileName</code>
<code>broadcast</code>	<code>postProcessor</code>
<code>calledProcedure</code>	<code>precondition</code>
<code>jobStep*</code>	<code>procedureName</code>

combinedStatus	projectName
command	propertySheetId
condition	releaseExclusive
createTime	releaseMode
delayUntil	resourceName
elapsedTime	resourceWaitTime
errorCode	retries
errorHandling	runAsUser
errorMessage	runnable
exclusive	runTime
exclusiveMode	shell
exitCode	start
external	status
finish	stepName
hostName	subprocedure
jobId	subproject
jobName	timeLimit
jobStepId	timeout
lastModifiedBy	totalWaitTime
licenseWaitTime	waitTime
liveProcedure	workingDirectory
liveProcedureStep	workspaceName
logFileName	workspaceWaitTime
modifyTime	

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	resourceName
exclusiveJobName	shell
exclusiveJobStepId	stepCount
exclusiveJobStepName	stepLimit
gateways	trusted
hostName	useSSL
hostOS	workspaceName
hostPlatform	zoneName
lastModifiedBy	

resourcePool

Contains elements to define a resource pool.

Contents:

- autoDelete
- createTime
- description
- lastModifiedBy
- lastResourceUsed
- modifyTime
- orderingFilter
- owner
- propertySheetId
- resourceNames
- resourcePoolDisabled
- resourcePoolId
- resourcePoolName

resourceUsage

Contains information about resource usage. For any step running on a resource, there is a resource usage record containing the ID and name of the job, job step, and resource.

Contents:

- jobId
- jobName
- jobStepId

jobStepName
licenceWaitTime
resourceId
resourceName
resourcePoolId
resourcePoolName
resourceUsageId
resourceWaitTime
waitReason
workspaceWaitTime

schedule

Contains all elements to define a schedule.

Contents:

actualParameters	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	postLogFileName
alwaysRun	postProcessor
attachedCredentials	precondition
attachedParameters	procedureName
broadcast	projectName
command	propertySheetId
condition	releaseExclusive
createTime	releaseMode
credentialName*	resourceName
description	shell
errorHandling	stepId
exclusive	stepName
exclusiveMode	subprocedure

<code>lastModifiedBy</code>	<code>subproject</code>
<code>logFileName</code>	<code>timeLimit</code>
<code>modifyTime</code>	<code>timeLimitUnits</code>
<code>owner</code>	<code>workingDirectory</code>
<code>parallel</code>	<code>workspaceName</code>

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
<code>aclEntryId</code>	number	The unique ElectricFlow-generated ID for this <code>aclEntry</code> object.
<code>active</code>	boolean	<i><Boolean flag - 0 1 true false></i> —If set to "true", the state of the workflow is active.
<code>activeCalls</code>	subcontainer	A container element within the <code>serverStatus</code> element. <code>activeCall</code> describes an API currently running on the server.
<code>activeState</code>	string	The name of the <code>activeState</code> on the workflow object.
<code>actualParameters</code>	propertySheet	An <code>actualParameter</code> 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.
<code>actualParameterId</code>	number	The unique ElectricFlow-generated ID for this actual parameter object.
<code>actualParameterName</code>	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.
<code>admin</code>	boolean	<i><Boolean flag - 0 1 true false></i> —If set to "true", the this is an "admin" license.
<code>adminLicenseLastUse</code>	date	The time at which the admin license was last used.
<code>adminLicenseUser</code>	string	The name of the user who is currently licensed as the "admin" user.
<code>agentDrivePath</code>	string	Drive-letter-based path used by Windows agents to access the workspace in steps.
<code>agentUncPath</code>	string	UNC path used by Windows ElectricFlow Web servers to access the workspace. The agent uses <code>agentUncPath</code> and <code>agentDrivePath</code> to compute the drive mapping needed for making <code>agentDrivePath</code> valid in the step.
<code>agentUnixPath</code>	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)
changePermissionsPrivilege	enum	Possible values are: allow deny inherit
code	enum	Script to execute the functions for a step—passed to the step's shell for execution.
combinedStatus	enum	More inclusive step status output - this value may combine up to three sub-elements: status message properties
command	string	The command to run steps - for command steps.
commonGroupNameAttribute	string	The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider.
completed	boolean	<Boolean flag - 0 1 true false> - If "true", the workflow is completed and no additional transactions will be evaluated.
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
<code>condition</code>	string	<p>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.</p> <p>For email notifiers: Mail sent if the condition evaluates to "true". The <code>condition</code> is a string subject to property expansion. The notification will NOT be sent if the expanded string is "false" or "0". If no <code>condition</code> is specified, the notification is ALWAYS sent.</p>
<code>configName</code>	string	The name of the configuration.
<code>container</code>	string	<p>An object ID for a "container" that contains formal parameters.</p> <p>In another context, this is typically the type and name of the workflow or job with a corresponding ID.</p>
<code>containerName</code>	string	The name of the container.
<code>createTime</code>	date	The time when this object was created.
<code>credentialId</code>	number	The unique ElectricFlow-generated ID for this credential object.
<code>credentialName</code>	string	<p><code>credentialName</code> can be one of two forms:</p> <p>relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object. Requires a qualifying project name.</p> <p>absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object's project.</p>
<code>customDatabaseDialect</code>	string	Class name for the Hibernate dialect. The server chooses an appropriate dialect based on <code>databaseType</code> or this can be part of the custom specification.
<code>customDatabaseDriver</code>	string	Class name of the JDBC driver. The server will choose an appropriate driver based on <code>databaseType</code> or this can be part of the custom specification.
<code>customDatabaseUrl</code>	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 <code><html> ... </html></code> tags. The only HTML tags allowed in the text are: <code><a></code> <code></code> <code>
</code> <code><div></code> <code><dl></code> <code></code> <code><i></code> <code></code> <code></code> <code><p></code> <code><pre></code> <code></code> <code><style></code> <code><table></code> <code><tc></code> <code><td></code> <code><th></code> <code><tr></code> <code></code>
destinations	string	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.
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	<p>Determines what happens to the procedure if the step fails:</p> <ul style="list-style-type: none"> • <code>failProcedure</code> - The current procedure continues, but the overall status is error (default). • <code>abortProcedure</code> - Aborts the current procedure, but allows already-running steps in the current procedure to complete. • <code>abortProcedureNow</code> - Aborts the current procedure and terminates running steps in the current procedure. • <code>abortJob</code> - Aborts the entire job, terminates running steps, but allows <code>alwaysRun</code> steps to run. • <code>abortJobNow</code> - Aborts the entire job and terminates all running steps, including <code>alwaysRun</code> steps. • <code>ignore</code> - 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	<p>Possible values are: <code>onCompletion onStart</code> "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".</p>
exclusive	boolean	<i><Boolean flag - 0 1 true false></i> - 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 <code>testDirectoryProvider</code> API, this element provides information on which groups the user is a member.
findUserTest	subcontainer	For the <code>testDirectoryProvider</code> 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
<code>formattingTemplate</code>	string	Specifies a template for formatting email messages when an event [notification] is triggered by the <code>emailNotifier</code> .
<code>fullUserName</code>	string	The user's full name - not his or her nickname.
<code>fullUserNameAttribute</code>	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.
<code>gatewayDisabled</code>	boolean	<i><Boolean flag -0 1 true false></i> If "true", the gateway is disabled.
<code>gatewayId</code>	number	The ElectricFlow-generated ID number for this gateway.
<code>gatewayName</code>	string	The name of the gateway.
<code>gateways</code>	list	A space-separated list of gateway names.
<code>gracePeriod</code>	number	The number of days available after the ElectricFlow license expires.
<code>groupBase</code>	string	This string is prepended to the <code>basedn</code> to construct the directory DN that contains group records.
<code>groupId</code>	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.
<code>groupList</code>	list	For the <code>testDirectoryProvider</code> API, this element contains zero or more groups returned after querying existing groups known to the directory provider.
<code>groupMemberAttributes</code>	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 <code>groupOfNames</code> or <code>uniqueGroupOfNames</code> records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: " <code>{0}</code> " is replaced with the full user record DN, and " <code>{1}</code> " 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 <i>Agent Host Name</i> 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 <i>Agent Host Name</i> 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, HP-UX, 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
<code>interval</code>	string	The repeat interval for starting new jobs.
<code>intervalUnits</code>	enum	Possible values are: <code>hours</code> <code>minutes</code> <code>seconds</code> <code>continuous</code> If set to <code>continuous</code> , ElectricFlow creates a new job as soon as the previous job completes.
<code>inUseHosts</code>	number	The number of hosts (agents) currently in use.
<code>inUseLicenses</code>	number	The number of user licenses currently in use.
<code>inUseProxiedHosts</code>	number	The number of proxy target hosts currently in use.
<code>jobId</code>	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.
<code>jobName</code>	string	The name of the job.
<code>jobNameTemplate</code>	string	Template used to determine the default name of jobs launched from a procedure.
<code>jobStepId</code>	number	The unique identifier for a job step, assigned automatically when the job step is created.
<code>jobStepName</code>	string	The name of the job step.
<code>label</code>	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.
<code>lastMessage</code>	string	Element returned by the <code>serverStatus</code> API showing the last message the server received.
<code>lastModifiedBy</code>	string	Shows who (generally a user name) last modified the object.
<code>lastResourceUsed</code>	string	The name of the most recently used resource from the pool.
<code>lastRunTime</code>	date	The last time a job was launched by a schedule. -or- In a <code>resource</code> response, this is the most recent time that a job step ran on the resource.
<code>lastUse</code>		Returned element in the <code>concurrentUsers</code> subcontainer (for the <code>licenseUsage</code> 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
<code>misfirePolicy</code>	enum	Possible values are: <code>ignore</code> <code>run once</code> 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 <code>'ignore'</code> , otherwise it will run the job immediately. Defaults to <code>"ignore"</code> .
<code>modifyPrivilege</code>	enum	Possible values are: <code>allow</code> <code>deny</code> <code>inherit</code>
<code>modifyTime</code>	date	The time when the object was last modified.
<code>monthDays</code>	string	Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space.
<code>mostActiveCalls</code>	number	The number of most active API calls since server startup.
<code>mutable</code>	boolean	If <code>"true,"</code> the member list of this group is editable within ElectricFlow via the web UI or the <code>modifyGroup</code> API.
<code>name</code>	string	The name of the directory provider.
<code>notifierName</code>	string	The name of the email notifier.
<code>objectId</code>	number	An object identifier returned by <code>findObjects</code> and <code>getObjects</code> . 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.
<code>objectName</code>	string	The name of the object.
<code>objectType</code>	enum	The type of object being described, for example: <code>project</code> , <code>procedure</code> , <code>step</code> , and so on.
<code>orderingFilter</code>	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	<p>Possible values for <code>outcome</code>:</p> <p>Note: The <code>outcome</code> is accurate only if the job status is "completed."</p> <p><code>success</code> - The job finished successfully.</p> <p><code>warning</code> - The job completed with no errors, but encountered some suspicious conditions.</p> <p><code>error</code> - The job has finished execution with errors.</p>
owner	string	The person (user name) who created the object.
parallel	boolean	<i><Boolean flag - 0 1 true false></i> - If set, indicates this step should run at the same time as adjacent steps marked to run as parallel also. Defaults to "false".
password	string	The password matching the specified user name.
path	string	The property path that specifies the object to use.
pingToken	number	Every time an agent starts, a unique <code>pingToken</code> value is generated. The server uses the <code>pingToken</code> 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	<p>If a port number is not specified, the default ElectricFlow port is used.</p> <p>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 <code>ssh</code>, the default is 22.</p>
port1	number	The port number used by <i>Gateway Resource1</i> - default is to the port number used by the resource.

Returned element	Type	Description/Value
port2	number	The port number used by <i>Gateway Resource2</i> - default is to the port number used by the resource.
position	number	Used to reorder a <i>ElectricFlow</i> 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. <i>ElectricFlow</i> includes a customizable program called "postp" for this purpose. The value for <code>postProcessor</code> is a command string for invoking a post-processor program in the platform shell for the resource (<code>cmd</code> for Windows, <code>sh</code> 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 <i>precondition</i> is evaluated. A <i>precondition</i> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a <code>\0</code> or <code>\false</code> 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: <code>group user</code>
priority	enum	Possible values are: <code>low normal high highest</code> 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	string	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 <code>ssh</code> . (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: <code>allow deny inherit</code>
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 <code>serverStatus</code> API - a call no longer active (completed). The API monitor keeps track of the 10 most recent calls.
releaseExclusive	boolean	<i><Boolean flag - 0 1 true false></i> Declares whether or not this step will release its resource, which is currently held exclusively.
releaseMode	string	Possible values are: <code>none release releaseToJob</code>
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	<i><Boolean flag - 0 1 true false></i> 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 <code>jobName</code> , <code>jobId</code> , and/or a <code>jobStepId</code> 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: <code>bootstrap</code> , <code>databaseConfiguration</code> , <code>databaseConnection</code> , <code>databaseSchema</code> , <code>running</code> , <code>failed</code> , <code>stopping</code> , <code>importFailed</code>
severity	enum	Possible values are: <code>INFO</code> <code>WARN</code> <code>ERROR</code>
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 " <code>cmd /q /c</code> " for a Windows agent and " <code>sh -e</code> " 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 <code>hh:mm</code> , 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 <code>status</code> : <p><code>pending</code> - The job is not yet runnable—it is waiting for other steps to complete first.</p> <p><code>runnable</code> - The job is ready to run, but it is waiting for a resource to become available.</p> <p><code>running</code> - The job is assigned to a resource and is executing the step command.</p> <p><code>completed</code> - The job finished executing.</p>
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 <code>hh:mm</code> , using the 24-hour clock. ElectricFlow stops creating new jobs at this time, but a job in progress will continue to run. If <code>stopTime</code> is not specified, ElectricFlow creates one job only on each specified day.
subject	string	Refers to the object the event concerns (similar to <code>container</code>).

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, <code>command</code> or <code>commandFile</code> 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	string	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: <code>success</code> <code>skipped</code> <code>failure</code>
time	date	The time of day to invoke this schedule's procedure (24-hour clock, for example, 17:00). For a <code>logEntry</code> response, <code>time</code> indicates the time at which data was written to the log.
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: <code>hours</code> <code>minutes</code> <code>seconds</code>
timeout	number	Specifies the timeout for the <code>element</code> 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	<p><Boolean flag - 0 1 true false> If "true", the resource is <i>trusted</i>. A trusted agent is one that has been "certificate verified."</p> <p>Agents can be either <i>trusted</i> or <i>untrusted</i>:</p> <ul style="list-style-type: none"> • 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	<p>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).</p> <p>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/.</p>

Returned element	Type	Description/Value
<code>userAuthenticationTest</code>	subcontainer	For the <code>testDirectoryProvider</code> API, this element authenticates the user.
<code>userBase</code>	string	The string prepended to the <code>basedn</code> to construct the directory DN that contains user records.
<code>userId</code>	number	The unique ElectricFlow-generated ID for the user.
<code>userInfo</code>		<code>findUserTest</code> container element includes a <code>userList</code> subcontainer that may include multiple <code>userInfo</code> tags, each of which describes a user (including full name, email address, and provider name).
<code>userList</code>	list	<code>findUserTest</code> container element includes a <code>userList</code> subcontainer that may include one or more <code>userInfo</code> tags.
<code>userName</code>	string	The full name of the user. For Active Directory and LDAP, the name may be <code>user@domain</code> .
<code>userNameAttribute</code>	string	The attribute in a user record that contains the user's account name.
<code>userSearchFilter</code>	string	The LDAP query performed in the context of the user directory to search for a user by account name. The string " <code>{0}</code> " is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.
<code>userSearchSubtree</code>	boolean	<i><Boolean flag - 0 1 true false></i> If true, the subtree below the user base was recursively searched.
<code>userSettingsId</code>	number	The unique ElectricFlow-generated ID for the user settings.
<code>useSSL</code>	boolean	<i><Boolean flag - 0 1 true false></i> 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.
<code>value</code>	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 manage artifacts.
- Create and manage object properties.
- Create and manage resources.
- Create workflows and add resources to them.
- Create and call procedures.
- Model and deploy applications.
- Model and run pipelines.

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.

tierMap		Show/Hide	List Operations	Expand Operations	Raw
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": "",
  "environmentName": ""
}

```

Response Content Type
application/json

Parameters

Parameter	Value	Description	Parameter Type	Data Type
projectName	<input data-bbox="474 877 803 905" type="text" value="{required}"/>	Name for the project; must be unique among all projects.	path	string
applicationName	<input data-bbox="474 947 803 974" type="text" value="{required}"/>	The name of the application	path	string
environmentName	<input data-bbox="474 999 803 1026" type="text" value="{required}"/>	The name of the environment.	query	string
environmentProjectName	<input data-bbox="474 1050 803 1077" type="text" value="{required}"/>	The name of the environment's project name.	query	string
tierMapName	<input data-bbox="474 1119 803 1146" type="text"/>	The name of the tier map. If not specified the operation will assume a hyphenated application and environment name.	query	string
tierMappings	<input data-bbox="474 1220 803 1247" type="text"/>	The list of mappings between the application tiers and the environment tiers.	query	string
body	<div data-bbox="474 1304 803 1423" style="border: 1px solid #ccc; height: 50px;"></div>	Use to submit JSON object with parameters, e.g. {"description": "My description"}	body	string

Parameter content type:
application/json

Try it out!

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": "",
  "environmentName": ""
}
```

Response Content Type application/json ▼

Parameters

Parameter	Value	Description	Parameter Type	Data Type
projectName	<input type="text" value="default"/>	Name for the project; must be unique among all projects.	path	string
applicationName	<input type="text" value="App1"/>	The name of the application	path	string
environmentName	<input type="text" value="env1"/>	The name of the environment.	query	string
environmentProjectName	<input type="text" value="default"/>	The name of the environment's project name.	query	string
tierMapName	<input type="text"/>	The name of the tier map. If not specified the operation will assume a hyphenated application and environment name.	query	string
tierMappings	<input type="text"/>	The list of mappings between the application tiers and the environment tiers.	query	string
body	<div><pre>{ "parameters":{ "tierMapping":[{ "environmentTier":"EnvTier2", </pre></div> <div>Parameter content type: application/json ▼</div>	Use to submit JSON object with parameters, e.g. {"description":"My description"}	body	string

Try it out!

4. Click **Try it out!** to run the resource.



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": ""
}

```

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?applicationName=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%20Mundane%20Application/processes?processName=AppProc1&timeLimit=30&timeLimitUnits=seconds`

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.

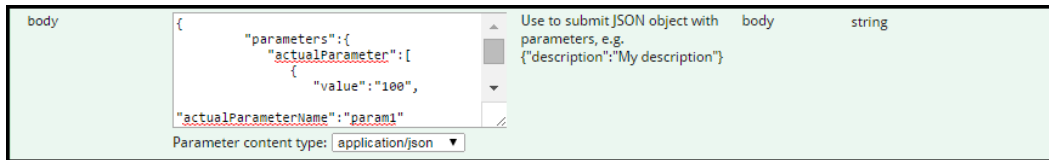
To make a schedule with actual parameters, enter the values in the parameter section as shown below:

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"
      }
    ]
  }
}
```



The response is this Request URL:

`https://localhost-100.electric-cloud.com/rest/v1.0/projects/default/schedules?scheduleName=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 DSL is an easy to use domain specific language that allows you to perform the same ElectricFlow operations that can be performed using the RESTful or Perl API.

- Create and manage artifacts.
- Create and manage object properties.
- Create and manage resources.
- Create workflows and add resources to them.
- Create and call procedures.
- Model and deploy applications.
- Model and run pipelines.

The ElectricFlow DSL is the easiest of the ElectricFlow programming constructs to use. You create scripts (templates) 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 ElectricFlow DSL is based on Groovy scripting language; however, you do not need to know Groovy in detail to use the DSL.

These are the benefits of using DSL scripts:

- You can create one master script (template) 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 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.

The following sections describe how to create DSL scripts (templates) and run them in background without using any API commands or requests.

Getting Started with DSL

[Creating and Running DSL Scripts](#) on page 798

[Common Use Cases](#) on page 798

[Generating a DSL Script for an Existing ElectricFlow Object](#) on page 799

[Passing Parameters or Arguments to Your DSL Script](#) on page 800

[Passing DSL Parameters or Arguments from the Command Line](#) on page 800

[Evaluating Your DSL Script in a Specific Application Context](#) on page 800

[Adding Support for Augmenting the Groovy Runtime classpath for ectool evalDsl](#) on page 801

[Debugging Your DSL Script](#) on page 802

Creating and Running DSL Scripts

Using ectool

To create and run a DSL script using `ectool`:

1. Use any text editor or Groovy IDE to create a script using ElectricFlow DSL methods, and save it.
2. Log into `ectool`.
3. Enter `evalDsl` on page 708 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. Start `ec-perl`.

If you used `ec-perl` in for the previous step, skip this step.

3. Enter `evalDsl` on page 708 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

Getting Help on DSL Methods

You can get information on the supported DSL methods by using `evalDsl` for help while creating your DSL script.

These options are available to describe DSL methods:

- 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
```

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, 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 allows you can 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 instances of ElectricFlow objects. For example, you have the following script to create a resource that uses SSL in the *secure* zone:

```
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 values can be passed to the DSL script using the following command:

```
ectool evalDsl  
--dslFile myScript.dsl  
--parameters '{"resourceName":"MyFirstResource", "resourceIP":"192.168.10.12"}'
```

Passing DSL Parameters or Arguments from the Command Line

ElectricFlow DSL also allows you to pass DSL parameters from the command line .

For example, to create a project, you can enter:

```
ectool evalDsl 'project(args.name)' --parameters '{"name":"baaz"}'
```

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 already exists  
    application('Deploy world') {  
        //'Web Tier' application tier will be created within 'Deploy world' application unless 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: 'DeployUtiliti  
es')
```

See the [ElectricFlow Perl API Commands](#) on page 51 for the complete list of API commands.

Adding Support for Augmenting the Groovy Runtime classpath for `ectool evalDsl`

Using External .jar Files With DSL

ElectricFlow DSL is based on Groovy so that you can take advantage of all the 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 *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 ->
        assert json.size() == 3
        json.responseData.results.each {
            result.put(it.titleNoFormatting, it.visibleUrl)
        }
    }
}
```

```
}  
  
result
```

2. Create a directory, `/opt/dslSamples/lib`, that should be 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 like any other class.

Debugging Your DSL Script

You can use the `debug` argument to generate debug output for your script as it is being evaluated by ElectricFlow DSL engine. The generated output is useful when debugging your DSL scripts.

```
ectool evalDsl --dslFile myScript.dsl --debug 1
```

or

```
ectool evalDsl <dsl text> --debug 1
```

DSL Methods

ElectricFlow supports these DSL methods:

Name	Description
<code>acl</code>	An access control list that controls access to domain objects.
<code>aclEntry</code>	An individual access control list entry (ACE) that allows or denies a privilege on a domain object.
<code>actualParameter</code>	A name/value pair that is passed to a procedure when it is invoked.
<code>application</code>	An ElectricFlow object that you model and deploy to build, test , deploy, and release your software for continuous delivery.
<code>applicationTier</code>	A logical grouping of a components that are part of an application and the resources on which they should be deployed.

Name	Description
<code>artifact</code>	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.
<code>artifactVersion</code>	An artifact version is a collection of 0 to N files that were published to an artifact repository.
<code>component</code>	An object that is based on a specific version of an artifact and is defined in an application.
<code>credential</code>	<p>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.</p> <p>It is usually used in an agent context to authenticate with a third-party system.</p> <p>The password value is not available in the web context for security reasons, so this object has no <code>getPassword</code> method.</p>
<code>deployerApplication</code>	An application that will be deployed as part of a Deployer in a Release
<code>deployerConfiguration</code>	The configuration for an application that will be deployed in a Release
<code>directoryProvider</code>	Contains information about the configuration used to communicate with an external directory service (LDAP or ActiveDirectory).
<code>emailConfig</code>	Encapsulates all of the mail server configuration information necessary for the ElectricFlow server to send an e-mail message.
<code>emailNotifier</code>	E-mail notification to be sent when a particular event occurs.
<code>environment</code>	The environment to which an application is deployed.
<code>environmentTemplate</code>	A template defining an environment that can be spun up when the application is deployed.
<code>environmentTemplateTier</code>	A logical grouping of resources in an environment template.
<code>environmentTemplateTierMap</code>	A map that contains the mapping of application tiers to the corresponding environment template tiers.
<code>environmentTier</code>	A logical grouping of resources in an environment.
<code>formalParameter</code>	An unbound parameter defined on a procedure, workflow definition, and so on.
<code>gateway</code>	A secure connection between two zones for sharing or transferring information between the zones.

Name	Description
<code>group</code>	A group of users.
<code>hook</code>	<p>A resource template hook that stores a reference to a procedure in an ElectricFlow project or plugin project.</p> <p>When a resource template is used to create a resource pool, these procedures are invoked.</p>
<code>job</code>	An instance of a procedure run.
<code>jobStep</code>	A step in a job.
<code>license</code>	License data in XML format that describes the usage to which you are entitled.
<code>note</code>	Notes for a Release.
<code>pipeline</code>	<p>An ElectricFlow object that orchestrates a deployment or automation.</p> <p>ElectricFlow supports these types of pipelines:</p> <ul style="list-style-type: none">• Release pipeline—Only for an application release.• Generic pipeline—For any deployment or automation.
<code>plugin</code>	An add-on program used by ElectricFlow to integrate with third-party tools, custom dashboards, and unique user experiences based on roles.
<code>procedure</code>	A container for steps that execute a task.
<code>process</code>	An application or component process.
<code>processDependency</code>	A dependency between process steps.
<code>processStep</code>	A step in an application or component process.
<code>project</code>	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.
<code>property</code>	<p>A <code>name-value</code> 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.</p> <p>ElectricFlow supports intrinsic and custom properties.</p>
<code>release</code>	A ElectricFlow object defined by a pipeline and a Deployer task.

Name	Description
<code>repository</code>	An object that stores artifact versions. It primarily contains information about how to connect to a particular artifact repository.
<code>resource</code>	An agent machine that is configured to communicate with ElectricFlow and where job steps can be executed.
<code>resourcePool</code>	A collection of resources with an ordering policy.
<code>resourceTemplate</code>	A template with the required information to provision and later spin up cloud resources on an on-demand basis.
<code>schedule</code>	An object that launches a procedure at a specified time in the future, possibly on a regular interval.
<code>snapshot</code>	A version of an application with specific artifact versions and at a specific state at any point in time.
<code>stage</code>	A logical grouping of pipeline tasks.
<code>stateDefinition</code>	A state definition in a workflow definition. Each workflow can contain one or more states.
<code>step</code>	A unit of logic that will execute on an agent.
<code>task</code>	A representation of task in a stage or gate.
<code>tierMap</code>	A map that contains the mapping of application tiers to the corresponding environment tiers.
<code>transitionDefinition</code>	How a workflow transitions from one state to another.
<code>user</code>	A user defines an account used to log into the system and control access to ElectricFlow objects.
<code>workflowDefinition</code>	Workflow objects are split into two types: <i>Definition</i> objects and <i>Instance</i> objects. Definition objects provide the template for a running workflow instance.
<code>workspace</code>	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.
<code>zone</code>	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
<code>applicationName</code>	The name of the application container of the property sheet that owns the property.
<code>applicationTierName</code>	The name of the application tier container of the property sheet that owns the property.
<code>artifactName</code>	The name of the artifact container of the property sheet that owns the property.
<code>artifactVersionName</code>	The name of the <code>artifactVersion</code> container of the property sheet that owns the property.
<code>componentName</code>	The name of the component container of the property sheet that owns the property.
<code>configName</code>	The name of the emailConfig container that owns the property.
<code>credentialName</code>	The name of the credential container of the property sheet that owns the property.
<code>environmentName</code>	The name of the environment container of the property sheet that owns the property.
<code>environmentTemplateName</code>	The name of the environment template container of the property sheet that owns the property.
<code>environmentTemplateTierName</code>	The name of the environment template tier container of the property sheet that owns the property.
<code>environmentTierName</code>	The name of the environment tier container of the property sheet that owns the property.
<code>executePrivilege</code>	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.
<code>flowName</code>	The name of the flow container of the property sheet that owns the property.

Name	Description
<code>flowRuntimeName</code>	The name of the flow runtime container of the property sheet that owns the property.
<code>flowRuntimeStateName</code>	The name of the flow runtime state container of the property sheet that owns the property.
<code>flowStateName</code>	The name of the flow state container of the property sheet that owns the property.
<code>flowTransitionName</code>	The name of the flow transition container of the property sheet that owns the property.
<code>gatewayName</code>	The name of the gateway container of the property sheet.
<code>groupName</code>	The name of the group container of the property sheet that owns the property.
<code>jobId</code>	The primary key or name of the job container of the property sheet that owns the property.
<code>jobStepId</code>	The primary key of the job-step container of the property sheet that owns the property.
<code>notifierName</code>	The name of the notifier container of the property sheet that owns the property.
<code>objectId</code>	The object id as returned by <code>findObjects</code> .
<code>path</code>	Property path string.
<code>pipelineName</code>	The name of the pipeline container of the property sheet that owns the property.
<code>pluginName</code>	The name of the plugin container of the property sheet that owns the property.
<code>procedureName</code>	The name of the procedure container of the property sheet that owns the property.
<code>processName</code>	The name of the process, if the container is a process or process step.
<code>processStepName</code>	The name of the process step, if the container is a process step.
<code>projectName</code>	The name of the project container of the property sheet that owns the property.
<code>propertySheetId</code>	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
<code>workflowName</code>	The name of the workflow container of the property sheet that owns the property.
<code>workspaceName</code>	The name of the workspace container of the property sheet.
<code>zoneName</code>	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
<code>principalName</code>	Name of the user or group for this access control entry.
<code>principalType</code>	Type of principal for this access control entry (user or group).

Optional Arguments

Name	Description
<code>applicationName</code>	The name of the application container of the property sheet that owns the property.
<code>applicationTierName</code>	The name of the application tier container of the property sheet that owns the property.
<code>artifactName</code>	The name of the artifact container of the property sheet that owns the property.
<code>artifactVersionName</code>	The name of the <code>artifactVersion</code> container of the property sheet that owns the property.
<code>changePermissionsPrivilege</code>	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
<code>modifyPrivilege</code>	Determines whether the principal can change the contents of the object.
<code>notifierName</code>	The name of the notifier container of the property sheet that owns the property.
<code>objectId</code>	The object id as returned by <code>findObjects</code> .
<code>path</code>	Property path string.
<code>pipelineName</code>	The name of the pipeline container of the property sheet that owns the property.
<code>pluginName</code>	The name of the plugin container of the property sheet that owns the property.
<code>procedureName</code>	The name of the procedure container of the property sheet that owns the property.
<code>processName</code>	The name of the process, if the container is a process or process step.
<code>processStepName</code>	The name of the process step, if the container is a process step.
<code>projectName</code>	The name of the project container of the property sheet that owns the property.
<code>propertySheetId</code>	The primary key of the property sheet that owns the property.
<code>readPrivilege</code>	Determines whether the principal can examine the contents of the object.
<code>releaseName</code>	The name of the Release container of the property sheet that owns the property.
<code>repositoryName</code>	The name of the repository container of the property sheet that owns the property.
<code>resourceName</code>	The name of the resource container of the property sheet that owns the property.
<code>resourcePoolName</code>	The name of the resource pool container of the property sheet that owns the property.
<code>resourceTemplateName</code>	The name of the resource template container of the property sheet that owns the property.
<code>scheduleName</code>	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
<code>actualParameterName</code>	The name of the parameter to create, modify, or delete.
<code>projectName</code>	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>applicationName</code>	The name of the application, if the actual parameter is on an application process step.
<code>componentName</code>	The name of the component, if the actual parameter is on a component process step.
<code>flowName</code>	The name of the flow to which the flow state belongs.
<code>flowStateName</code>	The name of the flow state, if the formal parameter is in a flow state.
<code>newName</code>	New name for an existing object that is being renamed.
<code>procedureName</code>	The name of the procedure.
<code>processName</code>	The name of the process, if the actual parameter is on a process step.
<code>processStepName</code>	The name of the process step, if the actual parameter is on a process step.
<code>releaseName</code>	The name of the Release container of the property sheet that owns the property.
<code>scheduleName</code>	The name of the schedule.
<code>stateDefinitionName</code>	The name of the state definition.
<code>stepName</code>	The name of the step.
<code>transitionDefinitionName</code>	The name of the state definition.
<code>value</code>	The value of the actual parameter, if creating or modifying.
<code>workflowDefinitionName</code>	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
<code>applicationName</code>	The name of the application.
<code>projectName</code>	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	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 <code>artifactKey</code> 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 <code>groupId</code> 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 <code>artifactKey</code> component of the <code>GroupId/ArtifactKey/Version</code> (GAV) coordinates.
artifactName	The name of the artifact containing the <code>artifactVersion</code> .
artifactVersionName	The name of the <code>artifactVersion</code> .
artifactVersionState	The state of the <code>artifactVersion</code> .
dependentArtifactVersions	<p>The set of <code>artifactVersions</code> on which this <code>artifactVersion</code> depends.</p> <p>An alternate name for this argument is <code>dependentArtifactVersion</code>.</p>
description	Comment text describing this object that is not interpreted at all by ElectricFlow.
groupId	The <code>groupId</code> component of the GAV (<code>GroupId/ArtifactVersionId/Version</code>) 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 <code>true</code> or <code>1</code> , all dependencies will be removed.
repositoryName	The name of the artifact repository.
version	The version component of the GAV (<code>GroupId/ArtifactVersionId/Version</code>) 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
<code>componentName</code>	The name of the component
<code>projectName</code>	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>actualParameters</code>	Parameters passed as arguments to the application component. The alternate argument name is <code>actualParameter</code> .
<code>applicationName</code>	The name of an application to which the component is scoped.
<code>clearActualParameters</code>	If this is set to <code>true</code> or <code>1</code> , the step removes all actual parameters.
<code>credentialName</code>	The name of a credential to attach to this component.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	The new name for an existing object.
<code>pluginKey</code>	The key of the plugin.
<code>pluginName</code>	The name of the plugin.
<code>pluginParameters</code>	List of plugin parameters. An alternate name is <code>pluginParameter</code> .
<code>sourceApplicationName</code>	The name of source application.
<code>sourceComponentName</code>	The name of new component.
<code>sourceProjectName</code>	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
<code>credentialName</code>	The name of the credential.
<code>projectName</code>	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	New name for an existing object.
<code>password</code>	The password for the credential. It can also be a certificate or a chunk of data.
<code>passwordRecoveryAllowed</code>	If set to <code>true</code> or <code>1</code> , ElectricFlow recovers the password by invoking <code>getFullCredential</code> from a job step.
<code>userName</code>	The user name 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
<code>applicationName</code>	The name of the application.
<code>projectName</code>	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>applicationProjectName</code>	The name of the project containing specified application. If it is not specified, the default is the Release project name.
<code>orderIndex</code>	The order in which the applications are deployed, starting from 1).
<code>processName</code>	The name of the application process.
<code>releaseName</code>	The name of the Release.
<code>smartDeploy</code>	Smart deploy flag used during the <code>runProcess</code> method for the application.
<code>snapshotName</code>	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
<code>applicationName</code>	The name of the application.
<code>projectName</code>	The name for the project that must be unique among all projects.
<code>stageName</code>	The name of the stage of a pipeline attached to a Release if a Release is specified.

Optional Arguments

Name	Description
<code>actualParameters</code>	Parameters passed as arguments to the application component. The alternate argument name is <code>actualParameter</code> .
<code>applicationProjectName</code>	The name of the project containing specified application. If it is not specified, the default is the Release project name.
<code>clearActualParameters</code>	If this is set to <code>true</code> or <code>1</code> , the step removes all actual parameters.
<code>environmentName</code>	The name of the environment.
<code>environmentProjectName</code>	The name of the project containing the specified environment or environment template. If this is not specified, the default is Release project name.
<code>environmentTemplateName</code>	The name of the environment template.
<code>environmentTemplateProjectName</code>	The name of the project containing specified environment template. If this is not specified, the default is the environment project name.
<code>environmentTierName</code>	The name of the environment tier container of the property sheet that owns the property.
<code>releaseName</code>	The name of the Release.
<code>skipDeploy</code>	If set to <code>true</code> or <code>1</code> , 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
<code>providerName</code>	Name for a LDAP directory provider that must be unique.

Optional Arguments

Name	Description
<code>commonGroupNameAttribute</code>	<p>The attribute in a group record that contains the common name of the group.</p> <p>If specified, it is only used when searching for groups from an external provider.</p> <p>It is usually used when the group name attribute is set to <code>distinguishedName</code>, because this field is not searchable.</p>
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>domainName</code>	The domain from which the Active Directory servers are automatically discovered.
<code>emailAttribute</code>	The attribute in a LDAP user record that contains the user's email.
<code>enableGroups</code>	Determines whether or not to enable external groups for the directory provider.
<code>fullUserNameAttribute</code>	The attribute in a user record that contains the user's full name (first and last).
<code>groupBase</code>	String prepended to the base distinguished name (DN) to construct the DN of the directory that contains group records.
<code>groupMemberAttributes</code>	Comma separated list of attribute names that can identify a member of a group.
<code>groupMemberFilter</code>	LDAP query string for the groups directory to find groups that contain a given user as a member.
<code>groupNameAttribute</code>	The attribute in a group record that contains the name of the group.
<code>groupSearchFilter</code>	LDAP query string used in group directory to enumerate group records.
<code>managerDn</code>	<p>The name of a user who has read-only access to the LDAP or Active Directory server. It is usually a distinguished name (DN).</p> <p>A simple name may be used when the Active Directory server URL is being auto-discovered via DNS.</p>
<code>managerPassword</code>	Secret value used to identify the account for the query user.
<code>newName</code>	New name for an existing object that is being renamed.
<code>providerType</code>	Type string for a directory provider: ldap or activedirectory .

Name	Description
<code>realm</code>	The realm of the LDAP directory provider. This is used to create unique user names when there are multiple providers.
<code>url</code>	The URL of the LDAP Directory Provider server.
<code>useSSL</code>	If this argument is set to <code>true</code> or <code>1</code> , SSL is used for communication.
<code>userBase</code>	Used to construct the distinguished name (DN) of the directory that contain user records.
<code>userNameAttribute</code>	The attribute in a user record that contains the user's account name.
<code>userSearchFilter</code>	RFC 2254 LDAP query to search for a user by name.
<code>userSearchSubtree</code>	If this argument is set to <code>true</code> or <code>1</code> , 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
<code>configName</code>	The email configuration name.

Optional Arguments

Name	Description
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>mailFrom</code>	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
<code>configName</code>	The name for an email configuration, or text that through property expansion results in such an email configuration name.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>destinations</code>	A list of space-separated user names, email addresses, or email aliases or text that through property expansion results in such a list.
<code>environmentNames</code>	Name of the environments. An alternate argument name is <code>environmentName</code> .
<code>eventType</code>	An enumeration of valid event categories recognized by the email notifiers.
<code>flowName</code>	The name of the flow container of the email notifier.
<code>flowStateName</code>	The name of the flow container of the email notifier.
<code>formattingTemplate</code>	String containing the email formatting instructions for generating notifications.
<code>gateType</code>	The type of the gate.
<code>groupNames</code>	A list of the groups that receives the notification. An alternate argument name is <code>groupName</code> .
<code>jobId</code>	The primary key or name of the job container of the email notifier.
<code>jobStepId</code>	The primary key of the job-step container of the email notifier.
<code>newName</code>	New name for an existing object.
<code>notificationType</code>	The notification type that will be stored in the <code>ec_notificationType</code> property.
<code>pipelineName</code>	The name of the pipeline container of the email notifier.
<code>procedureName</code>	The name of the procedure container of the email notifier.
<code>processName</code>	The name of the process container of the email notifier.
<code>processStepName</code>	The name of the process step container of the email notifier.
<code>projectName</code>	The name of the project container of the email notifier.
<code>stageName</code>	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 <code>userName</code> .
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 <code>true</code> or <code>1</code> , 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
<code>environmentTemplateName</code>	The name of the environment template.
<code>projectName</code>	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	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
<code>tierMapName</code>	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.
<code>tierMappings</code>	The list of mappings between the application tiers and the environment template tiers. An alternate argument name is <code>tierMapping</code> .

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
<code>environmentName</code>	The name of the environment.
<code>environmentTierName</code>	Name for the environment tier that must be unique among all tiers for the environment.
<code>projectName</code>	Name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	The new name for an object.
<code>resourceNames</code>	A list of resources to add to the environment tier. An alternate argument name is <code>resourceName</code> .

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
<code>formalParameterName</code>	The name for this parameter that is used when the procedure is invoked to specify a value for the parameter.
<code>projectName</code>	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>applicationName</code>	The name of the application, if the formal parameter is on an application process.
<code>componentName</code>	The name of the component, if the formal parameter is on a component process.
<code>defaultValue</code>	If no value is provided for the parameter when the procedure is invoked, this value will be used.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>expansionDeferred</code>	If this argument is set to <code>true</code> or <code>1</code> , 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.
<code>flowName</code>	The name of the flow to which the flow state belongs.
<code>flowStateName</code>	The name of the flow state if the formal parameter is on a flow state.
<code>label</code>	The display label.
<code>newName</code>	The new name for an existing object.
<code>orderIndex</code>	The display order index, starting from 1.
<code>pipelineName</code>	The name of the pipeline, if the formal parameter is on a pipeline.
<code>procedureName</code>	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 <code>true</code> or <code>1</code> , 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 <code>true</code> or <code>1</code> , the artifact repository is disabled.
hostName1	The domain name or IP address <code>resourceName2</code> uses to send messages to <code>resourceName1</code> .
hostName2	The domain name or IP address <code>resourceName1</code> uses to send messages to <code>resourceName2</code> .
newName	The new name for an existing object.

Name	Description
port1	The port number that <code>resourceName2</code> uses to send messages to <code>resourceName1</code> .
port2	The port number <code>resourceName1</code> uses to send messages to <code>resourceName2</code> .
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 <code>resourceName2</code> '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 <code>resourceName1</code> '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 <code>true</code> or <code>1</code> , all users are removed from this group.
userNames	The list of users in the group. An alternate argument name is <code>userName</code> .

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 <code>hookParameter</code> .
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
<code>destinationProject</code>	The project that will own the job.
<code>jobId</code>	The primary key of the job or the name of the job.
<code>jobNameTemplate</code>	The template used to determine the default name of jobs launched from a procedure.
<code>procedureName</code>	The name of the procedure that should own the job step. If this is not specified, <code>myStep.procedure</code> is used.
<code>projectName</code>	The name of the project if the <code>destinationProject</code> is not specified.
<code>status</code>	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
<code>actualParameters</code>	Actual parameters passed to an invoked subprocedure. An alternate argument name is <code>actualParameter</code> .

Name	Description
<code>alwaysRun</code>	If this argument is set to <code>true</code> or <code>1</code> , this step will run even if the preceding steps fail in a way that aborts the job.
<code>broadcast</code>	If this argument is set to <code>true</code> or <code>1</code> , 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).
<code>command</code>	The script to execute the functions of this step. It is passed to the step's shell for execution.
<code>condition</code>	A fixed text or text embedding property references that is evaluated into a logical <code>TRUE</code> or <code>FALSE</code> . An empty string, a <code>"0"</code> or <code>"false"</code> is interpreted as <code>FALSE</code> . Any other result string is interpreted as <code>TRUE</code> .
<code>credentialName</code>	The name of the credential to be used for impersonation.
<code>credentials</code>	The list of run-time credentials attached to the job step. An alternate argument name is <code>credential</code> .
<code>errorHandling</code>	Error handling method for this step.
<code>exclusive</code>	<p>If this argument is set to <code>true</code> or <code>1</code>, the resource acquired for this step will be retained for the exclusive use of this job. This means the following:</p> <ul style="list-style-type: none"> • 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.
<code>exclusiveMode</code>	<p>Determines the mode to use when the step acquires a resource.</p> <ul style="list-style-type: none"> • If set to <code>none</code>, the default behavior for the step applies. • If set to <code>job</code>, the resource will be retained for the exclusive use of this job. • If set to <code>step</code>, the resource will be retained for the exclusive use of this step and procedure it may call. • If set to <code>call</code>, the resource will be retained for the exclusive use of all steps within the current procedure call.
<code>external</code>	If this argument is set to <code>true</code> or <code>1</code> , the step is externally managed (no state machine).
<code>jobStepId</code>	The primary key of the job step

Name	Description
<code>jobStepName</code>	The name for the new step. If this is not specified, a default name will be generated.
<code>logFileName</code>	The name of the log file for a step that is specified relative to the root directory in the job's workspace.
<code>parallel</code>	If this argument is set to <code>true</code> or <code>1</code> , this step and all adjacent steps with the flag set will run in parallel.
<code>parentPath</code>	The path to the parent job step. If this is not specified, the current job step is used.
<code>postProcessor</code>	This command runs in parallel with the main command for the step. It analyzes the log for the step and collects diagnostic information.
<code>precondition</code>	A fixed text or text embedding property references that is evaluated into a logical <code>TRUE</code> or <code>FALSE</code> . An empty string, a <code>"0"</code> or <code>"false"</code> is interpreted as <code>FALSE</code> . Any other result string is interpreted as <code>TRUE</code> .
<code>procedureName</code>	The name of the procedure that should own the job step. If this is not specified, <code>myStep.procedure</code> is used.
<code>projectName</code>	The name of the project for the procedure.
<code>releaseExclusive</code>	If this argument is set to <code>true</code> or <code>1</code> , the resource acquired for this step will be no longer be retained for the exclusive use of this job when this step completes.
<code>releaseMode</code>	Determines the mode to use when the step releases its resource. <ul style="list-style-type: none"> • If set to <code>none</code>, the default behavior applies. • If set to <code>release</code>, the resource will be available for use by any job. • If set to <code>releaseToJob</code>, the resource will be available for use by any step in this job.
<code>resourceName</code>	The name for the resource that must be unique among all resources.
<code>shell</code>	The name of the shell program that will execute the command and postprocessor for the step.
<code>status</code>	The starting status for the step.
<code>stepName</code>	The name of the procedure step that should 'own' the job step. If not specified, <code>myStep</code> is used.

Name	Description
<code>subprocedure</code>	The name of a procedure to invoke during this step.
<code>subproject</code>	The name of the project containing the procedure to invoke during this step.
<code>timeLimit</code>	The maximum amount of time the step can execute; abort if it exceeds this time.
<code>timeLimitUnits</code>	The units for step time limit: <code>seconds</code> , <code>minutes</code> , or <code>hours</code> .
<code>workingDirectory</code>	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.
<code>workspaceName</code>	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
<code>licenseData</code>	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
<code>newName</code>	The new name for an existing object.
<code>note</code>	Object representing the notes for an entity.
<code>noteName</code>	The name of the note.
<code>projectName</code>	The project name of the entity that owns the note.
<code>releaseName</code>	The name of the Release.

pipeline

Creates or modifies a pipeline.

Required Arguments

Name	Description
<code>pipelineName</code>	The name of the pipeline
<code>projectName</code>	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>enabled</code>	If this argument is set to <code>true</code> or <code>1</code> , the pipeline is enabled.
<code>newName</code>	The new name for an existing object.
<code>type</code>	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
<code>procedureName</code>	The name for the procedure that must be unique within the project.
<code>projectName</code>	The name for the project that must be unique among all projects.

Optional Arguments

Name	Description
<code>credentialName</code>	The name of a credential to attach to this procedure.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>jobNameTemplate</code>	The template used to determine the default name of jobs launched from a procedure.
<code>newName</code>	The new name for an existing object.
<code>resourceName</code>	The name of the default resource for this procedure.
<code>timeLimit</code>	The maximum amount of time the step can execute. The step is aborted if it exceeds this time.
<code>timeLimitUnits</code>	The units for step time limit: <code>seconds</code> , <code>minutes</code> , or <code>hours</code> .
<code>workspaceName</code>	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: <code>seconds</code> , <code>minutes</code> , or <code>hours</code> .
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
<code>actualParameters</code>	Actual parameters passed to an invoked subprocedure or process. An alternate argument name is <code>actualParameter</code> .
<code>afterProcessStep</code>	If specified, the process step will be placed after the named process step.
<code>applicationName</code>	The name of the application if the process is owned by an application.
<code>applicationTierName</code>	If the step references an application tier, this is the name of the application tier.
<code>beforeProcessStep</code>	If specified, the process step will be placed before the named process step.
<code>clearActualParameters</code>	If this argument is set to <code>true</code> or <code>1</code> , the step will remove all actual parameters.
<code>componentApplicationName</code>	If specified, the component is scoped to this application, not the project.
<code>componentName</code>	The name of the component if the process is owned by a component.
<code>credentialName</code>	The name of the credential.
<code>dependencyJoinType</code>	The join type for incoming dependencies.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>errorHandling</code>	Error handling method for this step.
<code>includeCompParameterRef</code>	If this argument is set to <code>true</code> or <code>1</code> , the actual parameters will be generated from the component properties. This works only for artifact components.
<code>newName</code>	The new name for an existing object.
<code>processStepType</code>	The type of the process step
<code>subcomponent</code>	If the step referencing a component process, the name of the component.
<code>subcomponentApplicationName</code>	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
<code>tracked</code>	If this argument is set to <code>true</code> or <code>1</code> , Change Tracking is enabled for this project.
<code>workspaceName</code>	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
<code>propertyName</code>	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 <code>emailConfig</code> 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 <code>findObjects</code> .
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
<code>workflowName</code>	The name of the workflow container of the property sheet that owns the property.
<code>workspaceName</code>	The name of the workspace container of the property sheet.
<code>zoneName</code>	The name of the zone container of the property sheet.

release

Creates and modifies a Release.

Required Arguments

Name	Description
<code>projectName</code>	The name for the project that must be unique among all projects.
<code>releaseName</code>	The name of the Release.

Optional Arguments

Name	Description
<code>actualParameters</code>	Parameters passed as arguments to the application. The alternate argument name is <code>actualParameter</code> .
<code>clearActualParameters</code>	If this is set to <code>true</code> or <code>1</code> , the task removes all actual parameters.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	The name of the pipeline.
<code>pipelineName</code>	Whether or not to disable the repository.
<code>pipelineProjectName</code>	The name of the project containing the specified pipeline. If this is not specified, the default is the Release project name.
<code>plannedEndDate</code>	The date when this release is expected to end (for example, 2006-05-15).
<code>plannedStartDate</code>	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
<code>repositoryName</code>	The name of the repository.

Optional Arguments

Name	Description
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	New name for an existing object.
<code>repositoryDisabled</code>	Whether to disable the repository.
<code>url</code>	The URL for contacting the repository.
<code>zoneName</code>	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
<code>resourceName</code>	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 <code>true</code> or <code>1</code> , 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 <code>none</code> .
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 <code>none</code> .
repositoryNames	A newline delimited list of repositories from which to retrieve artifacts.
resourceDisabled	If this is set to <code>true</code> or <code>1</code> , 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
<code>trusted</code>	If this is set to <code>true</code> or <code>1</code> , the agent can speak to all other trusted agents in its zone. An untrusted agent can only speak to gateway agents.
<code>useSSL</code>	If this is set to <code>true</code> or <code>1</code> , SSL is used for communication.
<code>workspaceName</code>	The name of the workspace.
<code>zoneName</code>	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
<code>resourcePoolName</code>	The name for the resource pool that must be unique among all resource pools.

Optional Arguments

Name	Description
<code>autoDelete</code>	If this is set to <code>true</code> or <code>1</code> , the pool is deleted when the last resource is deleted.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	The new name for an existing object that is being renamed.
<code>orderingFilter</code>	The JavaScript fragment that returns custom ordering of resources in a pool.
<code>resourceNames</code>	The list of resources to add or remove from the pool. An alternate argument name is <code>resourceName</code> .
<code>resourcePoolDisabled</code>	If this is set to <code>true</code> or <code>1</code> , 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
<code>projectName</code>	Name for the project that must be unique among all projects.
<code>resourceTemplateName</code>	Name for the resource template that must be unique among all resource templates.

Optional Arguments

Name	Description
<code>cfgMgrParameters</code>	Configuration manager plugin parameters. An alternate argument name is <code>cfgMgrParameter</code> .
<code>cfgMgrPluginKey</code>	Configuration manager plugin key.
<code>cfgMgrProcedure</code>	Configuration manager plugin method name.
<code>cfgMgrProjectName</code>	Configuration manager plugin project name.
<code>cloudProviderParameters</code>	Cloud provider plugin parameters. An alternate argument name is <code>cloudProviderParameter</code> .
<code>cloudProviderPluginKey</code>	Cloud provider plugin key.
<code>cloudProviderProcedure</code>	Cloud provider plugin method name.
<code>cloudProviderProjectName</code>	Cloud provider plugin project name.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	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
<code>projectName</code>	Name for the project that must be unique among all projects.
<code>scheduleName</code>	Name for the schedule that must be unique among all schedules for the project.

Optional Arguments

Name	Description
<code>actualParameters</code>	Parameters passed to the invoked procedure, process, and/or workflow. An alternate argument name is <code>actualParameter</code> .
<code>applicationName</code>	The name of the application that owns the process.
<code>applicationProjectName</code>	The name for the project to which the application belongs.
<code>beginDate</code>	The date when this schedule will begin (for example, 2006-05-15).
<code>clearActualParameters</code>	Whether to clear actual parameters for this object.
<code>credentialName</code>	The name of the credential to use for impersonation.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>endDate</code>	The date when this schedule will end (for example, 2006-05-15). The end date is not included in the range of dates.
<code>environmentName</code>	The name of the environment used to determine where to run the process.
<code>environmentProjectName</code>	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 <code>tierResourceCount</code> .

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 <code>LATEST</code> to indicate latest version. The alternate argument name is <code>componentVersion</code> .
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
<code>projectName</code>	The name for the project that must be unique among all projects.
<code>stateDefinitionName</code>	The name used for the state definition
<code>workflowDefinitionName</code>	The name of the workflow definition.

Optional Arguments

Name	Description
<code>actualParameters</code>	The actual parameters to the state definition's process. An alternate argument name is <code>actualParameter</code> .
<code>clearActualParameters</code>	If this is set to <code>true</code> or <code>1</code> , the state definition removes all actual parameters for the process.
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	The new name for an existing object.
<code>startable</code>	If this is set to <code>true</code> or <code>1</code> , the workflow can begin in this state.
<code>subprocedure</code>	The name of the sub procedure.
<code>subproject</code>	The name of the project that contains the subprocedure.
<code>substartingState</code>	The name of the starting state to use in the <code>subworkflowDefinition</code> .
<code>subworkflowDefinition</code>	The name of the <code>subworkflowDefinition</code> .

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 <code>actualParameter</code> .
afterProcedureStep	If specified, the procedure step will be placed after the named procedure step.
alwaysRun	If this is set to <code>true</code> or <code>1</code> , 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 <code>true</code> or <code>1</code> , 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 <code>true</code> or <code>1</code> , 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 <code>TRUE</code> or <code>FALSE</code> . An empty string, a <code>"0"</code> or <code>"false"</code> is interpreted as <code>FALSE</code> . Any other result string is interpreted as <code>TRUE</code> .
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
<code>exclusive</code>	<p>If this is set to <code>true</code> or <code>1</code>, the resource acquired for this step will be retained for the exclusive use of this job. This means the following:</p> <ul style="list-style-type: none"> • 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.
<code>exclusiveMode</code>	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.
<code>logFileName</code>	The name of the log file for a step; specified relative to the root directory in the job's workspace.
<code>newName</code>	The new name for an existing object.
<code>parallel</code>	If this is set to <code>true</code> or <code>1</code> , this step and all adjacent steps with the flag set will run in parallel.
<code>postProcessor</code>	This command runs in parallel with the main command for the step; it analyzes the log for the step and collects diagnostic information.
<code>precondition</code>	A fixed text or text embedding property references that is evaluated into a logical TRUE or FALSE. An empty string, a <code>"0"</code> or <code>"false"</code> is interpreted as FALSE. Any other result string is interpreted as TRUE.
<code>releaseExclusive</code>	If this is set to <code>true</code> or <code>1</code> , the resource acquired for this step will be no longer be retained for the exclusive use of this job when this step completes.
<code>releaseMode</code>	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.
<code>resourceName</code>	Name for the resource that must be unique among all resources.
<code>shell</code>	Name of the shell program that will execute the command and postprocessor for the step.
<code>subprocedure</code>	Name of a procedure to invoke during this step.

Name	Description
<code>subproject</code>	Name of the project containing the procedure to invoke during this step.
<code>timeLimit</code>	Maximum amount of time the step can execute; abort if it exceeds this time.
<code>timeLimitUnits</code>	Units for step time limit: seconds, minutes, or hours.
<code>workingDirectory</code>	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.
<code>workspaceName</code>	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
<code>projectName</code>	The name for the project that must be unique among all projects.
<code>taskName</code>	The name of the task

Optional Arguments

Name	Description
<code>actualParameters</code>	Actual parameters passed to an invoked subprocedure. An alternate argument name is <code>actualParameter</code> .
<code>advancedMode</code>	If this is set to <code>true</code> or <code>1</code> , advanced mode is enabled so that a template can be created from a snapshot.
<code>afterTask</code>	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 <code>approver</code> .
beforeTask	If specified, the task will be placed before the named task.
clearActualParameters	If this is set to <code>true</code> or <code>1</code> , the task removes all actual parameters.
credentials	Credentials to be used in the task. An alternate argument name is <code>credential</code> .
description	Comment text describing this object that is not interpreted at all by ElectricFlow.
enabled	If this is set to <code>true</code> or <code>1</code> , 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 <code>true</code> or <code>1</code> , ElectricFlow keeps environment on error. The default is <code>false</code> .
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 <code>true</code> or <code>1</code> , 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 <code>tierResourceCount</code> .
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 <code>tierMapping</code> .

transitionDefinition

Defines how a workflow must transition from one state to another.

Required Arguments

Name	Description
<code>projectName</code>	Name for the project that must be unique among all projects.
<code>stateDefinitionName</code>	The name used for the state definition.
<code>transitionDefinitionName</code>	The name used for the transition definition.
<code>workflowDefinitionName</code>	The name of the workflow definition.

Optional Arguments

Name	Description
<code>actualParameters</code>	The actual parameters to the transition's target state. An alternate name is <code>actualParameter</code> .
<code>clearActualParameters</code>	If this is set to <code>true</code> or <code>1</code> , the transition should remove all actual parameters from the target state.
<code>condition</code>	A fixed text or text embedding property references that is evaluated into a logical <code>TRUE</code> or <code>FALSE</code> . If the string is empty or this is set to <code>false</code> or <code>0</code> , the text is interpreted as <code>FALSE</code> . If the string is not empty or this is set to <code>true</code> or <code>1</code> , the text is interpreted as <code>TRUE</code> .
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	New name for an existing object that is being renamed.
<code>targetState</code>	Target state for the transition definition.
<code>trigger</code>	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
<code>userName</code>	The user name.

Optional Arguments

Name	Description
<code>email</code>	Email address of the user.
<code>fullUserName</code>	Full name of the user.
<code>groupNames</code>	List of groups that this user is in. An alternate name is <code>groupName</code> .
<code>migrateSettings</code>	Migrate the user or group settings to this user name.
<code>newName</code>	New name for an object that is being renamed.
<code>password</code>	The user's password.
<code>removeFromAllGroups</code>	If this is set to <code>true</code> or <code>1</code> , ElectricFlow removes this user from all groups.
<code>sessionPassword</code>	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
<code>projectName</code>	Name for the project that must be unique among all projects.
<code>workflowDefinitionName</code>	The name of the workflow definition.

Optional Arguments

Name	Description
<code>description</code>	Comment text describing this object that is not interpreted at all by ElectricFlow.
<code>newName</code>	New name for an object that is being renamed.
<code>workflowNameTemplate</code>	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
<code>workspaceName</code>	The workspace name.

Optional Arguments

Name	Description
<code>agentDrivePath</code>	On Windows, the path name to the root directory of a workspace, specified with a drive letter.
<code>agentUncPath</code>	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	<p>To get the complete list of supported DSL methods, enter the following command:</p> <pre>ectool evalDsl --describe 1</pre> <p>To get help on a particular DSL method, enter the following command:</p> <pre>ectool evalDsl <dsl_method_name> --describe 1</pre> <p>Example: <code>ectool evalDsl procedure --describe 1</code></p>	<p>Getting Started with DSL on page 797</p> <p>Using the ElectricFlow Perl API on page 40.</p>
The DSL script completes successfully using the <code>evalDsl</code> command; however, the ElectricFlow objects are not created or updated as expected.	<p>Use the <code>debug</code> option to trace and debug the script processing using the following command:</p> <pre>ectool evalDsl <dsl> --debug 1</pre> <p>The <code>debug</code> option will allow <code>evalDsl</code> to generate debug output that you can use to follow the DSL script processing by <code>evalDsl</code> and identify any possible issue with the DSL script.</p>	<p>evalDsl on page 708</p> <p>Creating and Running DSL Scripts on page 798</p>

FAQs

1. 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.

2. 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.

3. 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`

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
```