ElectricFlow 6.1
API Guide

Electric Cloud, Inc.
www.electric-cloud.com
Copyright © 2002 – 2015 Electric Cloud, Inc. All rights reserved.

Published 2/17/2016

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.
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job Management</td>
<td>55</td>
</tr>
<tr>
<td>Parameter Management</td>
<td>57</td>
</tr>
<tr>
<td>Plugin Management</td>
<td>57</td>
</tr>
<tr>
<td>Procedure Management</td>
<td>58</td>
</tr>
<tr>
<td>Project Management</td>
<td>58</td>
</tr>
<tr>
<td>Property Management</td>
<td>59</td>
</tr>
<tr>
<td>Resource Management</td>
<td>59</td>
</tr>
<tr>
<td>Schedule Management</td>
<td>60</td>
</tr>
<tr>
<td>Server Management</td>
<td>61</td>
</tr>
<tr>
<td>User/Group Management</td>
<td>61</td>
</tr>
<tr>
<td>Workflow Definition Management</td>
<td>62</td>
</tr>
<tr>
<td>Workflow Management</td>
<td>63</td>
</tr>
<tr>
<td>Workspace Management</td>
<td>63</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>64</td>
</tr>
<tr>
<td>API Commands Listed in Alphabetical Order</td>
<td>64</td>
</tr>
</tbody>
</table>

- **API Commands - ACL Management** ............................................... 75
- **API Commands - Applications** .................................................. 115
- **API Commands - Application Tier** ............................................ 119
- **API Commands - Artifact Management** ......................................... 126
- **API Commands – Change History** ................................................ 161
- **API Commands - Component** ...................................................... 168
- **API Commands - Credential Management** ....................................... 179
- **API Commands - Database Configuration** ...................................... 190
- **API Commands - Directory Provider Management** .......................... 194
- **API Commands - Dynamic Environment** .......................................... 207
- **API Commands - Email Configuration Management** .......................... 249
- **API Commands - Email Notifier Management** .................................. 255
- **API Commands - Environment** ..................................................... 285
- **API Commands - Environment Tier** .............................................. 297
- **API Commands - Gateway and Zone Management** ................................ 303
- **API Commands - Job Management** ................................................ 313
- **External Job APIs** ...................................................................... 334
- **API Commands - Parameter Management** ........................................ 357
- **API Commands - Pipelines and Releases** ....................................... 388
- **API Commands - Plugin Management** .............................................. 443
- **API Commands - Procedure Management** ........................................ 451
- **API Commands - Process** .............................................................. 472
- **API Commands - Process Dependency** ............................................. 481
- **API Commands - Process Step** ...................................................... 487
- **API Commands - Project Management** ............................................ 496
- **API Commands - Property Management** .......................................... 502
- **API Commands - Resource Management** .......................................... 547
- **API Commands - Schedule Management** .......................................... 568
- **API Commands - Server Management** ............................................. 580
- **API Commands - Snapshots** ........................................................... 591
- **API Commands - Tier Map** ............................................................ 598
- **API Commands - User and Group Management** .................................. 603
Introduction to ElectricFlow

ElectricFlow™ (including the ElectricFlow platform, formerly known as ElectricCommander) is an end-to-end Continuous Delivery application suite. It accelerates the continuous delivery of software and makes software delivery processes more repeatable, visible, scalable, and efficient. It provides domain-specific capabilities to automate some or all phases of your software delivery process, including the build, test, integrate, deploy, and release processes.

ElectricFlow gives distributed DevOps teams shared control and visibility into infrastructure, tool chains, and processes. It accelerates and automates the software delivery process and enables agility, availability, predictability, and security across many build-test, deployment, and release pipelines.

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:
ElectricFlow

- **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 retrieves a specific artifact version from a local repository, guaranteeing a consistent package from build to build.

• Preflight build functionality–Used by developers to build and test code changes in isolation on their local machines before those changes are committed to a production build.

• Plugin capability–ElectricFlow is built with an extensible UI, enabling easy development of plugins that include integrations with other tools, custom dashboards, and unique user experiences based on roles. "Bundled" plugins, installed during ElectricFlow installation, provide easy integration with your SCM systems, defect tracking applications, and so on.

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

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:

<table>
<thead>
<tr>
<th>API Type</th>
<th>Go to</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSL Methods</td>
<td>Using the ElectricFlow DSL on page 760</td>
</tr>
<tr>
<td>RESTful APIs</td>
<td>Using the ElectricFlow RESTful API on page 747</td>
</tr>
<tr>
<td>Perl APIs</td>
<td>Using ectool on page 41</td>
</tr>
<tr>
<td></td>
<td>Using Perl (ec-perl) on page 42</td>
</tr>
<tr>
<td></td>
<td>Using API Commands in Javascript on page 50</td>
</tr>
</tbody>
</table>
# ElectricFlow Glossary

These objects and concepts apply to ElectricFlow.

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>access control</td>
<td>An access control list (ACL) determines if a particular user can perform a particular operation on a specified object. The list contains access control entries (ACE), each of which specifies a user or group and indicates whether certain operations are allowed or denied for that user or group. Using access control provides security for ElectricFlow system use. See the Access Control topic for more information.</td>
</tr>
<tr>
<td>access control entry</td>
<td>An access control list (ACL) determines if a particular user can perform a particular operation on a specified object. The list contains access control entries (ACE), each of which specifies a user or group and indicates whether certain operations are allowed or denied for that user or group. Using access control provides security for ElectricFlow system use. See the Access Control topic for more information.</td>
</tr>
<tr>
<td>access control list</td>
<td>An access control list (ACL) determines if a particular user can perform a particular operation on a specified object. The list contains access control entries (ACE), each of which specifies a user or group and indicates whether certain operations are allowed or denied for that user or group. Using access control provides security for ElectricFlow system use. See the Access Control topic for more information.</td>
</tr>
<tr>
<td>ACE</td>
<td>See access control entry on page 18.</td>
</tr>
<tr>
<td>ACL</td>
<td>See access control list on page 18.</td>
</tr>
<tr>
<td>actual parameter</td>
<td>An actual parameter is an object that provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from &quot;formal parameters&quot;: formal parameters define parameters a procedure is expecting, and actual parameters provide values to use at run-time.</td>
</tr>
<tr>
<td>admin</td>
<td>&quot;admin&quot; is a special built-in user that has universal ElectricFlow access. If you log in as admin, you can perform any operation in the system, regardless of access control limitations.</td>
</tr>
<tr>
<td>agent</td>
<td>An agent is an 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.</td>
</tr>
</tbody>
</table>

See the Web Interface Help > Resources topic for more information.
## Glossary

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API</strong></td>
<td>Application Programming Interface. ElectricFlow supports these APIs based on the higher-order to the lower-order programming constructs:</td>
</tr>
<tr>
<td></td>
<td>• DSL methods—You create scripts and templates without using API commands to execute the script.</td>
</tr>
<tr>
<td></td>
<td>• RESTful APIs—You go to the RESTful API URI and enter the appropriate information on the API UI to execute to request.</td>
</tr>
<tr>
<td></td>
<td>• Perl APIs—You access the command-line interface through ectool or ec-perl and enter the correct command syntax to execute the command.</td>
</tr>
<tr>
<td><strong>application</strong></td>
<td>Application can refer to one of these entities:</td>
</tr>
<tr>
<td></td>
<td>• Within ElectricFlow, you model and deploy (run) an application to build, test, deploy, and release your software for continuous delivery across different pipelines.</td>
</tr>
<tr>
<td></td>
<td>An application consists of application tiers, components in these tiers, component processes, and application processes.</td>
</tr>
<tr>
<td></td>
<td>Before running an application in ElectricFlow, you must map its application tiers to corresponding environment tiers where the application will run.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td>Application Inventory</td>
<td>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.</td>
</tr>
<tr>
<td>application process</td>
<td>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.</td>
</tr>
<tr>
<td>application tier</td>
<td>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.</td>
</tr>
<tr>
<td>artifact</td>
<td>An artifact is a top-level object containing artifact versions, a name template for published artifact versions, artifact specific properties, and access control entries to specify privileges.</td>
</tr>
<tr>
<td>artifact key</td>
<td>An artifact key is an identifier for an artifact and the &quot;key&quot; component of the artifact name.</td>
</tr>
<tr>
<td>artifact repository</td>
<td>See the Artifact Management &gt; Artifact objects &gt; Repository topic in the ElectricFlow User Guide for more information.</td>
</tr>
<tr>
<td>Object</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>artifact version</td>
<td>An artifact version is a collection of 0 to N files that were published to an artifact repository.</td>
</tr>
<tr>
<td>automation platform</td>
<td>See ElectricFlowplatform.</td>
</tr>
<tr>
<td>backingstore</td>
<td>The backingstore is the directory on the repository server where artifact versions are stored. By default, the backingstore is the &lt;datadir&gt;/repository-data directory in the repository installation—this default setting can be changed.</td>
</tr>
<tr>
<td>build-test automation</td>
<td>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).</td>
</tr>
<tr>
<td>change history</td>
<td>A record of the historical states of the system and the changes between them over time.</td>
</tr>
<tr>
<td>Change Tracking</td>
<td>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 change history</td>
</tr>
<tr>
<td>component</td>
<td>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. A component is the result of running an ElectricFlow application and has details, properties, and access control settings. 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.</td>
</tr>
<tr>
<td>component process</td>
<td>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.</td>
</tr>
<tr>
<td>compression</td>
<td>Compression reduces transfer time when publishing an artifact. However, compression also adds overhead when computing the compressed data. If files included in the artifact version are primarily text files or are another highly compressible file format, the benefit of reduced transfer time outweighs the cost of computing compressed data.</td>
</tr>
<tr>
<td>Continuous Delivery</td>
<td>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. The goal of Continuous Delivery is to keep software release-ready, and enable a repeatable, reliable way to deploy software to any environment.</td>
</tr>
<tr>
<td>Object</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Continuous</td>
<td>Using continuous integration means a build is launched every time code changes are checked into a Source Control Management (SCM) system. 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.</td>
</tr>
<tr>
<td>Integration</td>
<td></td>
</tr>
<tr>
<td>credential</td>
<td>A credential is an object that stores a user name and password for later use. You can use credentials for user impersonation and saving passwords for use inside steps. Two credential types are available: stored or dynamic.</td>
</tr>
<tr>
<td>custom property</td>
<td>Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner and behave in every way like an intrinsic object-level property with one exception: they are not created automatically when the object is created. Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution. Custom properties in a property sheet can be one of two types: string property or a property sheet property. String properties hold simple text values. Property sheet properties hold nested properties. Nested properties are accessed by way of the property sheet property of their containing sheet.</td>
</tr>
</tbody>
</table>
| deploy           | Deploy can refer to one of these activities:  
  - 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. 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.  
    **Note:** Within ElectricFlow, the terms *deploy* and *run* are synonymous. When deploying an application in ElectricFlow, you are actually running it to produce your software or application.  
  - 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. |
<table>
<thead>
<tr>
<th><strong>Object</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>deploying (running) applications</td>
<td>You can deploy applications one of these ways:</td>
</tr>
<tr>
<td></td>
<td>- Full deploy—The system deploys all the objects (including application processes, components, and artifacts) in the application.</td>
</tr>
<tr>
<td></td>
<td>- 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.</td>
</tr>
<tr>
<td></td>
<td>- Partial deploy—The system deploys only objects that you select.</td>
</tr>
<tr>
<td></td>
<td>- Partial deploy with specific artifact versions—The system deploys only the artifacts with selected versions.</td>
</tr>
<tr>
<td></td>
<td>- Schedule—Create schedules to run applications on a one-time, daily, weekly, or monthly basis.</td>
</tr>
<tr>
<td></td>
<td>- Snapshot—Select a snapshot to deploy.</td>
</tr>
<tr>
<td>deployment automation</td>
<td>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).</td>
</tr>
<tr>
<td>description</td>
<td>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>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td>diagnostic extract</td>
<td>A diagnostic extract is a log file portion from a job step, typically describing an error or interesting condition, extracted by a postprocessor and saved for reporting. The postprocessor usually places this information in an XML file in the top-level job workspace directory, and then sets a property that contains the filename.</td>
</tr>
<tr>
<td></td>
<td>The ElectricFlow postp postprocessor uses filenames like <code>diag-2770.xml</code>, where &quot;2770&quot; is the unique identifier for the step. Other postprocessors you may use can have a different filename configuration.</td>
</tr>
<tr>
<td>domain specific language</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>DSL</td>
<td>See <strong>domain specific language</strong> on page 22.</td>
</tr>
<tr>
<td>dynamic credential</td>
<td>Dynamic credentials are captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded.</td>
</tr>
<tr>
<td>Object</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| dynamic environment    | An environment that is automatically created on an on-demand basis, when you deploy an application in ElectricFlow.  
  - 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.  
  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). |
| 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                | *ec-perl* is a small wrapper program installed as part of ElectricFlow. When the ec-perl wrapper runs, it sets up the environment, finds, and calls the ElectricFlow copy of Perl, passing all of its parameters to Perl.  
  You can run ec-perl when using the ElectricFlow Perl API.                                                                                                         |
| ectool                 | *ectool* is the ElectricFlow command-line application that provides control over the ElectricFlow system if you prefer using a command-line interface rather than the ElectricFlow web interface. Most functions that can be invoked through the ElectricFlow web interface can be invoked using ectool.  
  You can run ectool when using the ElectricFlow Perl API.                                                                                                         |
| 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 *ElectricCommander*). |
| ElectricSentry         | ElectricSentry is the ElectricFlow engine for continuous integration—integrating with numerous Source Control Management (SCM) systems. ElectricSentry is installed automatically with ElectricFlow and is contained in a ElectricFlow plugin named ECSCM and in the Electric Cloud project.  
  **Note:** The CI Continuous Integration Dashboard is the front-end user interface for ElectricSentry.                                                                                     |
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>email configuration</td>
<td>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.</td>
</tr>
<tr>
<td>email notifier</td>
<td>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.</td>
</tr>
<tr>
<td>environment</td>
<td>Within the ElectricFlow system, the location to which a resource is assigned and where jobs, procedures, application, or pipelines run. See also dynamic environment and static environment. Before running an application in ElectricFlow, you must map its application tiers to corresponding environment tiers where the application will run.</td>
</tr>
<tr>
<td>Environment Inventory</td>
<td>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.</td>
</tr>
<tr>
<td>environment template</td>
<td>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.</td>
</tr>
<tr>
<td>environment tier</td>
<td>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.</td>
</tr>
<tr>
<td>event log</td>
<td>Log information about system events.</td>
</tr>
<tr>
<td></td>
<td>See log on page 27 for more information.</td>
</tr>
<tr>
<td>Everyone</td>
<td>A special intrinsic access control group that includes all users.</td>
</tr>
<tr>
<td>filter</td>
<td>Two filter categories:</td>
</tr>
<tr>
<td></td>
<td>* Intrinsic filters—these filters provide a convenient way to access certain well-defined fields for jobs.</td>
</tr>
<tr>
<td></td>
<td>* Custom filters—these filters allow you to access a much broader range of values, including custom properties. Any values accessible through an intrinsic filter can be checked using a custom filter also (though not as conveniently).</td>
</tr>
<tr>
<td>Object</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>formal parameter</td>
<td>A formal parameter is an object that defines a parameter expected by a procedure, including its name, a default value, and an indication of whether the parameter is required. Formal parameters are different from actual parameters. Formal parameters define the kinds of parameters a procedure is expecting, and actual parameters provide values to use at run-time.</td>
</tr>
<tr>
<td>gateway</td>
<td>To communicate with a resource, workspace, or artifact repository server in another zone, a gateway must be created. A gateway object contains two resource (agent) machines. For example, GatewayResource1 and GatewayResource2 are configured to communicate with the other. One gateway resource resides in the source zone and the other in the target zone. A gateway is bidirectional and informs the ElectricFlow server that each gateway machine is configured to communicate with a gateway machine in another zone.</td>
</tr>
<tr>
<td>group</td>
<td>A group defines a collection of users for access control purposes. A group can be defined externally in an LDAP or Active Directory repository, or locally in the ElectricFlow server. See the ElectricFlow Help &gt; Web Interface Help &gt; Users and Groups &gt; Groups topic for more information.</td>
</tr>
<tr>
<td>impersonation</td>
<td>Impersonation is a mechanism that allows a job step to execute under a particular login account (the ElectricFlow agent &quot;impersonates&quot; a particular user during the execution of that step). Impersonation is implemented using credentials.</td>
</tr>
<tr>
<td>inheritance</td>
<td>A feature of the 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.</td>
</tr>
<tr>
<td>intrinsic property</td>
<td>Intrinsic properties represent attributes that describe the object to which they are attached. ElectricFlow automatically provides intrinsic properties for each similar type object within ElectricFlow. For example: Every project has a description property that can be referenced with a non-local property path such as /projects/Examples/description.</td>
</tr>
<tr>
<td>Inventory Tracking</td>
<td>How ElectricFlow tracks what is deployed for continuous delivery. 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.</td>
</tr>
<tr>
<td>Object</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>job</td>
<td>A job is the output associated with invoking a ElectricFlow procedure. A new job is created each time you run (execute) a procedure. 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.</td>
</tr>
</tbody>
</table>
| job configuration   | A job configuration is an object containing all parameter and credential information needed to run a procedure. A Job Configuration section is provided as part of the 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:  
  - 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    | 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:  
  
  ```  
  ${[projectName]}_${[increment] /myproject/jobCounter}_${[timestamp]}  
  ```  
  
  which produces a name like:  
  
  `projectFoo_1234_20140102130321`  
  
  You can enter any combination of elements to create procedure names more meaningful to you. For example, you could choose to include the build number and procedure name. |
<p>| jobs quick view      | 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. |
| job step             | After a procedure is executed, the resulting job contains one job step for each step in the original procedure. The job step records information about the procedure step execution, such as the command executed, the resource where it executed, execution time, and error information. |
| job workspace        | A directory (containing all files and subdirectories) allocated by ElectricFlow for a particular job. Each job workspace is allocated as the child of a workspace root directory. |</p>
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>launch pad</td>
<td>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.</td>
</tr>
<tr>
<td>local group</td>
<td>A group defined inside ElectricFlow, as opposed to a group defined in an external repository. A local group can refer to both local and remote users, whereas a group in an external repository refers to users in that repository only. See group on page 25 for more information.</td>
</tr>
<tr>
<td>local user</td>
<td>A user defined inside ElectricFlow, as opposed to a user defined in an external repository. If a user defined in an external repository has the same name as a local user, the external user is not accessible. Local users are not visible outside ElectricFlow. Electric Cloud recommends using external accounts whenever available, but you may need to create local users if you do not have a shared directory service or if you need special accounts to use for ElectricFlow only. See user on page 38 for more information.</td>
</tr>
<tr>
<td>log</td>
<td>ElectricFlow provides a log for events generated anywhere in the system, including jobs and workflows.</td>
</tr>
</tbody>
</table>

**Note:** From the Administration tab, the default view for the Event Log page is the warning (WARN) level. For workflow and job event logs, the default view from their respective pages is the information (INFO) level.

- To see only events for a single workflow, select the Workflows tab, then a workflow Name to go to the Workflow Details page and click the View Log link at the top of the page.
- To see only events for a single job, select the Jobs tab, then the Job name to go to the Job Details page and click the View Log link at the top of the page.
- To see only events for a specific object, select the Search tab to go to the Define Search page. For example, you can select the Object Type, "Log Entry", 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 matcher 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. |

Glossary
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
</table>
| misfire policy  | A misfire policy allows you to manage how a schedule resumes in cases where the normal scheduled time is interrupted. Available options are:  
- `skip` (all misfires are ignored and the job runs at the next scheduled time).  
- `run once` (after one or more misfires, the job runs at the soonest time that occurs within an active region).  
  See schedule on page 35 for more information. |
| parameter       | A property value passed into a procedure when it is invoked (at run time), and used by the procedure to change its behavior. Two types of parameters are actual parameter on page 18 and formal parameter on page 25 |
| pipeline        | Pipelines easily model a reusable single pathway to production across multiple stages and environments and provide an aggregated view of all the activity.  
Use a pipeline when orchestrating a deployment consisting of more than one application or when orchestrating automations.  
Pipeline objects are split into two types: Definition objects and Instance objects.  
In this ElectricFlow release, you can model generic pipelines. |
| pipeline management | 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). |
| pipeline definition | Pipeline definition objects provide a template for a running pipeline instance.  
When you run the pipeline definition, ElectricFlow creates a new pipeline object that represents the run-time instance of the pipeline definition. |
| pipeline instance | Each time a pipeline runs, ElectricFlow creates an instance of the pipeline.  
For each instance, ElectricFlow creates logs of what happened. You can view these pipeline details:  
- Status  
- Time  
- Who  
- What  
- When  
- Evidence |
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plugin</td>
<td>An add-on program used by ElectricFlow to integrate with third-party tools, custom dashboards, and unique user experiences based on roles. 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. 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.</td>
</tr>
<tr>
<td>polling frequency</td>
<td>The polling frequency is how often the ElectricSentry continuous integration engine is set to look for new code check-ins. The default is set to every 5 minutes.</td>
</tr>
<tr>
<td>pool</td>
<td>Also known as resource pool. A pool is a collection of resources. If a step specifies a pool name as its resource, ElectricFlow can choose any available resource within that pool.</td>
</tr>
<tr>
<td>postp</td>
<td>postp is a postprocessor included with ElectricFlow. postp uses regular expression patterns to detect interesting lines in a step log. postp is already configured with patterns to handle many common cases such as error messages and warnings from gcc, gmake, cl, junit, and cppunit, or any error message containing the string &quot;error.&quot; postp also supports several useful command-line options, and it can be extended using &quot;matchers&quot; to handle environment-specific errors. See matcher on page 27 for more information.</td>
</tr>
<tr>
<td>postprocessor</td>
<td>A postprocessor is a command associated with a particular procedure step. After a step executes, the postprocessor runs to analyze its results. Typically, a postprocessor scans the step log file to check for errors and warnings. Also, it records useful metrics such as the number of errors in properties on the job step, and extracts step log portions that provide useful information for reporting. ElectricFlow includes a standard postprocessor called postp for your use and you can &quot;extend&quot; postp. See matcher on page 27 for more information.</td>
</tr>
<tr>
<td>preflight build</td>
<td>A preflight build provides a way to build and test a developer's changes before those changes are committed. A &quot;post-commit&quot; source tree is simulated by creating a clean source snapshot and overlaying the developer's changes on top of it. These sources are then passed through the production build procedure to validate the changes work successfully. Developers are allowed to commit their changes only if the preflight build is successful. Because developer changes are built and tested in isolation, many common reasons for broken production builds are eliminated.</td>
</tr>
<tr>
<td>Object</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>privileges</td>
<td>ElectricFlow supports four privilege types for access control on page 18 and security for each object:</td>
</tr>
<tr>
<td></td>
<td>• Read - Allows object contents to be viewed.</td>
</tr>
<tr>
<td></td>
<td>• Modify - Allows object contents (but not its permissions) to be changed.</td>
</tr>
<tr>
<td></td>
<td>• Execute - If an object is a procedure or it contains procedures (for example, a project), this privilege allows object procedures to be invoked as part of a job. For resource objects, this privilege determines who can use this resource in job steps.</td>
</tr>
<tr>
<td></td>
<td>• Change Permissions - Allows object permissions to be modified.</td>
</tr>
<tr>
<td>procedure</td>
<td>A procedure defines a process to automate one or more steps. A procedure is the ElectricFlow unit you execute (run) to carry out a process. A step in one procedure can call another procedure (in the same or different project), and this procedure then becomes known as a &quot;subprocedure&quot; (also known as a &quot;nested&quot; procedure). The step can pass arguments to the subprocedure.</td>
</tr>
<tr>
<td></td>
<td>During deployment automation, you can also call a procedure in an application or component process.</td>
</tr>
<tr>
<td>process branching</td>
<td>How to run job steps in an application or component process on a conditional basis in ElectricFlow.</td>
</tr>
<tr>
<td>process type</td>
<td>Select one of the following parameters to configure how Inventory Tracking works on an application or component process in ElectricFlow:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Deploy</strong>—Select this to enable Inventory Tracking. The ElectricFlow server tracks artifacts deployed to environments. This is the default.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Undeploy</strong>—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.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Other</strong>—Select this to disable Inventory Tracking.</td>
</tr>
<tr>
<td>procedure</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Object Description

<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
</table>
| project         | A project is a top-level container for related procedures, workflows, schedules, jobs, and properties, which is used to isolate different user groups or functions, and also encapsulate shared facilities. 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. Projects have two purposes:  
- Projects allow you to create separate work areas for different purposes or groups of people so they do not interfere with each other. In a small organization, you might choose to keep all work within a single project, but in a large organization, you may want to use projects to organize information and simplify management.  
- Projects simplify sharing. You can create library projects containing shared procedures and invoke these procedures from other projects. After creating a library project, you can easily copy it to other ElectricFlow servers to create uniform processes across your organization. |
<p>| project principal | Project principal is a special user ID associated with each project. If a project name is &quot;xyz,&quot; the project principal for that project is &quot;project: xyz&quot; (with an embedded space). This principal is used when procedures within the project are run, so you can create access control entries for this principal to control runtime behavior. |</p>
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>A property is a <strong>name-value</strong> 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: <strong>intrinsic</strong> and <strong>custom</strong>. ElectricFlow provides intrinsic properties and allows you to create custom properties. <strong>Note:</strong> Intrinsic properties are case-sensitive. Custom properties, like all other object names in the ElectricFlow system, are case-preserving, but not case-sensitive.</td>
</tr>
<tr>
<td>property sheet</td>
<td>A property sheet is a collection of properties that can be nested to any depth. The property value can be a string or a nested property sheet. Most objects have an associated &quot;property sheet&quot; that contains custom properties created by user scripts.</td>
</tr>
<tr>
<td>proxy agent</td>
<td>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.</td>
</tr>
<tr>
<td>proxy resource</td>
<td>This resource type requires SSH keys for authentication. You can create proxy resources (<strong>agent on page 18</strong> and target) for ElectricFlow to use on numerous other remote platforms/hosts that exist in your environment.</td>
</tr>
<tr>
<td>proxy target</td>
<td>A proxy target is an agent machine on an unsupported platform that can run commands via an SSH server.</td>
</tr>
<tr>
<td>publisher</td>
<td>A publisher is the job that completes the publish operation for an artifact version.</td>
</tr>
</tbody>
</table>

Intrinsic properties These properties represent attributes that describe the object to which they are attached, and are automatically by ElectricFlow for each similar type object.

For example, every project has a Description property that can be referenced with a non-local property path such as /projects/Examples/description.

- **Custom properties**
  Custom properties are identical to intrinsic properties and when placed on the same object, are referenced in the same manner, and behave in every way like an intrinsic object-level property with one exception: they are not created automatically when the object is created.

  Instead, custom properties can be added to objects already in the database before a job is started, or created dynamically by procedure steps during step execution.
### quiet time
An inactivity period before starting a build within a continuous integration system. This time period allows developers to make multiple, coordinated check-ins to ensure a build does not start with some of the changes only—assuming all changes are checked-in within the specified inactivity time period. This time period also gives developers an opportunity to "back-out" a change if they realize it is not correct. Using ElectricSentry, the inactivity time period can be configured globally for all projects or individually for a single project.

### RESTful API
An API for web services based on the REST (REpresentational State Transfer), a simple stateless architecture.

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.

### report
Reports display your information graphically for review or to show trends and other information.

ElectricFlow provides multiple reports and custom report capabilities to help you manage your build environment.

- 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:
  - 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
A repository is an object that stores artifact versions. This object primarily contains information about how to connect to a particular artifact repository. Similar to steps in a procedure, repository objects are in a user-specified order. When retrieving artifact versions, repositories are queried in this order until one containing the desired artifact version is found.

Connection information is stored in the repository object on the ElectricFlow server.

The artifact repository is a machine where artifact versions are stored in either uncompressed tar archives or compressed tar-gzip archives. The repository server is configured to store artifact versions in a directory referred to as the repository **backingstore**.

By default, the backingstore is the `<datadir>/repository-data directory in the repository installation.`
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
</table>
| resource            | An agent machine that is configured to communicate with ElectricFlow and where job steps can be executed. Steps can run on any resource or resource pool that you define. 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. ElectricFlow supports two types of resources:  
  - **Standard**—specifies a machine running the ElectricFlow agent on one of the supported agent platforms  
  - **Proxy**—requires SSH keys for authentication. You can create proxy resources (agents and targets) for ElectricFlow to use on numerous other remote platforms/hosts that exist in your environment.  
See also **dynamic resource** and **static resource**. During deployment automation, you define resources in environments during deployment automation. |
| resource template   | 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.                                                                                     |
| RESTful API         | A method of communication between a Web-based client and server that employs representational state transfer (REST) constraints.                                                                                                                                                                                                              |
| run                 | *Run* can refer to these activities:  
  - **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.  
    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.  
    **Note**: Within ElectricFlow, the terms *deploy* and *run* are synonymous. When deploying an application in ElectricFlow, you are actually running it to produce your software or application.  
  - **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. |
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
</table>
| schedule      | A schedule is an object used to execute procedures automatically in response to system events. For example, a schedule can specify executing a procedure at specific times on specific days. Three types of schedules are available:  
  - Standard—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      | 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.  
  
  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. |
| 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 state. Each state in a workflow, when it becomes active, can perform an action. A state can run a procedure to create a subjob or run a workflow definition to create a subworkflow—in the same way that procedures can call other procedures. One or more states can be designated as "starting" states to provide multiple entry points into the workflow. |
| state definition | Workflow objects are split into two types: Definition objects and Instance objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects.  
  
  When you run the workflow definition, the system creates a new workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition.  
  
  **Note:** We omit the "Instance" qualifier for brevity in the API and the UI.  
  
  Each workflow can contain one or more state objects. Defining states for a workflow is analogous to defining steps for a procedure. |
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static environment</td>
<td>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.</td>
</tr>
<tr>
<td>static resource</td>
<td>A resource located in your system or network, not in the cloud. Servers, databases, and agent machines are examples of static resources.</td>
</tr>
</tbody>
</table>
| step                   | 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. Each step specifies a command to execute on a particular resource or a subprocedure (nested procedure) to invoke. Commonly created steps include:  
  * Command—This step invokes a `bat`, `cmd`, `shell`, `perl` script, or similar.  
  * Subprocedure—This step invokes another ElectricFlow procedure.  
  * Plugin step—These include task-specific steps. Depending on which step-type you choose, the information you need to enter is somewhat different. Some of the step types bundled with ElectricFlow include:  
    * Publish or retrieve artifact version  
    * Send Email  
    * Various SCM step types  
    * Build tools                                                                                                                                                                                                 |
<p>| stored credential      | Stored credentials are given a name and stored in encrypted form in the database. Each project contains a list of stored credentials it owns. These credentials are managed from the Project Details page.                                                                                                                                 |
| subprocedure           | Creating subprocedures is a way of &quot;nesting&quot; procedures. A step (from any procedure) can call a procedure from another project or the same project. The procedure called by the step then becomes a subprocedure.                                                                                                                    |
| substitution           | A mechanism used to include property values in step commands and elsewhere. For example, if a step command is specified as &quot;<code>echo $[status]&quot;&quot;, and when the step executes there is a property named &quot;status&quot; with value &quot;success&quot;, the actual command executed will be &quot;</code>echo success&quot;. |
| system object          | This is a special object whose access control lists are used to control access to some ElectricFlow internals. 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>
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tag</td>
<td>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 &quot;mark&quot; a project to identify its relationship to one or more other projects or groups. For example, you might want to tag a group of projects as &quot;production&quot; or &quot;workflow&quot;, 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.</td>
</tr>
<tr>
<td>tier map</td>
<td>The mapping of the application tiers to the corresponding environment tiers where the application will run. To run an application, you map one application tier to one or more environment tiers and must have at least tier map.</td>
</tr>
</tbody>
</table>
| transition      | Transitions are used to move workflow progress from one state to another state. Four types of transitions are available to move a workflow to the next state:  
  - On Enter - transitions before sending notifiers or starting the sub-action  
  - On Start - transitions immediately after starting the sub-action. These transitions are ignored if no sub-action is specified for the source state.  
  - On Completion - transitions when the sub-action completes. These transitions are ignored if no sub-action is specified for the source state.  
  - Manual - transitions when a user selects the transition in the UI and specifies parameters. The same action can occur using ectl 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 | How a workflow transitions from one state to another.  
When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition.  
**Note:** We omit the "Instance" qualifier for brevity in the API and the UI.  
Each state can contain one or more transition objects. The transition definition object requires a name for the transition. This transition name will appear on the Workflow Definition Details page for quick reference and also on the State Definition Details page when you select the Transition Definitions tab.  
You can define one or more transitions for each state, depending on which transition options you want to apply to a particular state. |
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>A user defines an account used to log into the system and control access to ElectricFlow objects. A user can be defined externally in an LDAP or Active Directory repository, or locally in ElectricFlow. See <a href="#">local user</a> on page 27 for more information.</td>
</tr>
<tr>
<td>workflow</td>
<td>You can use a workflow to design and manage processes at a higher level than individual jobs. For example, workflows allow you to combine procedures into processes to create build-test-deploy lifecycles. A workflow contains states and transitions you define to provide complete control over your workflow process. The ElectricFlow Workflow feature allows you to define an unlimited range of large or small lifecycle combinations to meet your needs. Workflow objects are split into two types: Definition objects and Instance objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects.</td>
</tr>
<tr>
<td>workflow definition</td>
<td>Workflow objects are split into two types: Definition objects and Instance objects. Definition objects provide the template for a running workflow instance. You create a new workflow by defining a Workflow Definition along with its State Definition and Transition Definition objects. When you run the workflow definition, the system creates a new Workflow object with an equivalent set of State and Transition objects that represent the run-time instances of the workflow definition. Note: We omit the &quot;Instance&quot; qualifier for brevity in the API and the UI.</td>
</tr>
</tbody>
</table>
| workflow name template | This is the template used to determine the default name of jobs launched from the workflow definition. For example: 

```
${[projectName]}_${[increment]}/myproject/workflowCounter_${[timestamp]}
```

(substitute your values for the names above) Produces a workflow name like:

```
projectName_123_20140102130321
```
<table>
<thead>
<tr>
<th>Object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspace</td>
<td>A workspace is a subtree of files and directories where job file data is stored. The term &quot;workspace&quot; typically refers to the top-level directory in this subtree. 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. 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. 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. The location of the job step workspace is displayed on the Job Details page for the job under the &quot;Details&quot; for the step.</td>
</tr>
<tr>
<td>workspace root</td>
<td>A workspace root is a directory in which ElectricFlow allocates job workspace directories. Each workspace root has a logical name used to refer to it in steps and procedures.</td>
</tr>
<tr>
<td>zones</td>
<td>A zone is a way to partition a collection of agents to secure them from use by other groups. 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. A default zone is created during ElectricFlow installation. The server implicitly belongs to the default zone, which means all agents in this zone can communicate with the server directly (without the use of a gateway).</td>
</tr>
</tbody>
</table>
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
Using the ElectricFlow Perl API

- 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 bldg1server login "Ellen Ernst" "ee123"
```

General syntax for ectool command usage:

```
ectool [global argument] <command> <positional arguments> [named arguments]
```

Global Arguments (Optional)

See the Common global options section for more information.

Passing Lists as Arguments

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

For ectool

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

- name/value pairs - each pair is specified as a separate argument in the form name=value
  
  Example:
  ```
  ectool runProcedure proj1 --procedureName proc1 --actualParameter parml=value1 parm2=value2
  ```
**For ec-perl**

- **value list** - the argument value is a reference to an array of values
  
  Example:

  ```perl
  $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:

  ```perl
  $cmdr->runProcedure({
    projectName => 'proj1',
    procedureName => 'procl',
    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 = $cmdr->setProperty("filename", "temp.xml");
print "Return data from Commander:
".$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:

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

An alternative to using the `abortOnError` flag:

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

### Specifying a Named Object

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

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

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

### Common Global Options

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

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

**The Batch API**

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

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

Using the Batch API

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

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

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

Example - creating a batch object:

```perl
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:

```perl
# 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:

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

Sample response from this example:

```xml
<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>
</responses>
```
<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:

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

Single-transaction batch processing can continue after errors if you 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:
   ```perl
   my $batch = $N->newBatch('single', 'DuplicateResourceName');
   ```

2. Specify the ignoreErrors attribute as an argument to an individual request. In this case, the attribute applies only to that request and will override any global value specified:
   ```perl
   my $req2 = $batch->createResource($resource, {ignoreErrors => 'DuplicateResourceName'});
   ```

### Installing 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:

```bash
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:

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

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

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

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

  ```javascript
  ectool evalScript --value "JSON.stringify(api.getResources({}))" "{"resource":{"resourceId":"ceecf5-2d0d-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 4 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.electriccloud.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":"2014-08-26T10:50:23.7862","modifyTime":"2014-08-26T10:50:23.7862","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:

  ```javascript
  ectool evalScript --value "api.getResources({}).resource[0].resourceName"
  ```
ElectricFlow Perl API Commands

In this section, you can find Perl API commands one of these ways:

- **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.

- **Perl API Commands in Alphabetical Order**
  This section is one continuous list of API commands in alphabetical order.

From both lists, 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)

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

Artifact Management
<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addDependentsToArtifactVersion</td>
<td>Adds an artifact version query to an existing artifact. Dependent artifact versions are retrieved when the parent artifact version is retrieved.</td>
</tr>
<tr>
<td>cleanupArtifactCache</td>
<td>Deletes stale artifact versions from an artifact cache. A &quot;stale artifact version&quot; is one whose metadata was previously deleted from the ElectricFlow server.</td>
</tr>
<tr>
<td>cleanupRepository</td>
<td>Deletes stale artifact versions from the repository backing-store. A &quot;stale artifact version&quot; is one whose metadata was previously deleted from the ElectricFlow server.</td>
</tr>
<tr>
<td>createArtifact</td>
<td>Creates a new artifact.</td>
</tr>
<tr>
<td>createRepository</td>
<td>Creates a repository for one or more artifacts.</td>
</tr>
<tr>
<td>deleteArtifact</td>
<td>Deletes an existing artifact element and all artifact versions.</td>
</tr>
<tr>
<td>deleteArtifactVersion</td>
<td>Deletes artifact version metadata from the ElectricFlow database. (This API call does not delete or remove artifacts stored on the repository machine.)</td>
</tr>
<tr>
<td>deleteRepository</td>
<td>Deletes artifact repository metadata from the ElectricFlow database. (This API call does not delete or remove artifacts stored on the repository machine.)</td>
</tr>
<tr>
<td>findArtifactVersions</td>
<td>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 &quot;available&quot; state, and if run in a job step, registers the step as a retriever for the returned artifact versions. Use the Perl API for the findArtifactVersions command.</td>
</tr>
<tr>
<td>getArtifact</td>
<td>Retrieves an artifact by its name.</td>
</tr>
<tr>
<td>getArtifacts</td>
<td>Retrieves all artifacts in the system.</td>
</tr>
<tr>
<td>getArtifactVersion</td>
<td>Retrieves an artifact version by its name.</td>
</tr>
<tr>
<td>getArtifactVersions</td>
<td>Retrieves all artifact versions in the system, filtered by artifact name, retriever job ID, and/or retriever job step ID.</td>
</tr>
<tr>
<td>getManifest</td>
<td>Retrieves the manifest for a specified artifact version, which includes a list of files and directories in the artifact version, plus its checksum file.</td>
</tr>
<tr>
<td>getRepositories</td>
<td>Retrieves all artifact repository objects known to the ElectricFlow server.</td>
</tr>
<tr>
<td>getRepository</td>
<td>Retrieves an artifact repository by its name.</td>
</tr>
</tbody>
</table>


### ElectricFlow Perl API Commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifyArtifact</td>
<td>Modifies an existing artifact.</td>
</tr>
<tr>
<td>modifyArtifactVersion</td>
<td>Modifies an existing artifact version.</td>
</tr>
<tr>
<td>modifyRepository</td>
<td>Modifies an existing artifact repository.</td>
</tr>
<tr>
<td>moveRepository</td>
<td>Moves an artifact repository in front of another, specified repository or to the end of the list.</td>
</tr>
<tr>
<td>publishArtifactVersion</td>
<td>Publishes an artifact version to an artifact repository.</td>
</tr>
<tr>
<td>removeDependentsFromArtifactVersion</td>
<td>Removes a list of dependent artifact versions from an existing artifact version.</td>
</tr>
<tr>
<td>retrieveArtifactVersions</td>
<td>Retrieves the most recent artifact version, including its dependents, from an artifact repository.</td>
</tr>
</tbody>
</table>

### Credential Management

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

### Database Configuration

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDatabaseConfiguration</td>
<td>Retrieves the current database configuration.</td>
</tr>
</tbody>
</table>
Directory Provider Management

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

Email Configuration and Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createEmailConfig</td>
<td>Creates a new email configuration.</td>
</tr>
<tr>
<td>deleteEmailConfig</td>
<td>Deletes an email configuration.</td>
</tr>
<tr>
<td>getEmailConfig</td>
<td>Retrieves an email configuration by name.</td>
</tr>
<tr>
<td>getEmailConfigs</td>
<td>Retrieves all email configurations.</td>
</tr>
<tr>
<td>modifyEmailConfig</td>
<td>Modifies an existing email configuration.</td>
</tr>
</tbody>
</table>

Email Notifiers Management
### ElectricFlow Perl API Commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createEmailNotifier</td>
<td>Creates an email notifier on an object specified by an emailNotifierSelector.</td>
</tr>
<tr>
<td>deleteEmailNotifier</td>
<td>Deletes an email notifier from a property sheet container.</td>
</tr>
<tr>
<td>getEmailNotifier</td>
<td>Retrieves an email notifier from a property sheet container.</td>
</tr>
<tr>
<td>getEmailNotifiers</td>
<td>Retrieves all email notifiers defined for the specified property sheet container.</td>
</tr>
<tr>
<td>modifyEmailNotifier</td>
<td>Modifies an email notifier in a property sheet container specified by an emailNotifierSelector.</td>
</tr>
<tr>
<td>sendEmail</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Gateway and Zone Management

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

### Job Management
<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortAllJobs</td>
<td>Aborts all running jobs.</td>
</tr>
<tr>
<td>abortJob</td>
<td>Aborts a running job.</td>
</tr>
<tr>
<td>abortJobStep</td>
<td>Aborts any type of step—command step or subprocedure step.</td>
</tr>
<tr>
<td>completeJob</td>
<td>Completes an externally managed job.</td>
</tr>
<tr>
<td>completeJobStep</td>
<td>Completes an externally managed job step.</td>
</tr>
<tr>
<td>createJob</td>
<td>Creates an externally managed job.</td>
</tr>
<tr>
<td>createJobStep</td>
<td>Creates a job step in an existing job.</td>
</tr>
<tr>
<td>deleteJob</td>
<td>Deletes a job from the ElectricFlow database.</td>
</tr>
<tr>
<td>findJobSteps</td>
<td>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.</td>
</tr>
<tr>
<td>getJobDetails</td>
<td>Retrieves complete information about a job, including details from each job step.</td>
</tr>
<tr>
<td>getJobInfo</td>
<td>Retrieves all information about a job, except job step information.</td>
</tr>
<tr>
<td>getJobNotes</td>
<td>Retrieves the notes property sheet from a job.</td>
</tr>
<tr>
<td>getJobs</td>
<td>Retrieves summary information for a list of jobs.</td>
</tr>
<tr>
<td>getJobsForSchedule</td>
<td>Retrieves jobs started by a specific schedule.</td>
</tr>
<tr>
<td>getJobStatus</td>
<td>Retrieves the status of a job.</td>
</tr>
<tr>
<td>getJobStepDetails</td>
<td>Retrieves details for a job step.</td>
</tr>
<tr>
<td>getJobStepStatus</td>
<td>Retrieves the status of a job step.</td>
</tr>
<tr>
<td>modifyJob</td>
<td>Modifies the status of an externally managed job.</td>
</tr>
<tr>
<td>modifyJobStep</td>
<td>Modifies the status of an externally managed job step.</td>
</tr>
<tr>
<td>moveJobs</td>
<td>Moves jobs from one project to another.</td>
</tr>
<tr>
<td>runProcedure</td>
<td>Creates and starts a new job using a procedure directly or specified indirectly through a schedule.</td>
</tr>
<tr>
<td>setJobName</td>
<td>Sets the name of a running job.</td>
</tr>
</tbody>
</table>
Parameter Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachParameter</td>
<td>Attaches a formal parameter to a step.</td>
</tr>
<tr>
<td>createActualParameter</td>
<td>Creates a new actual parameter for a step that calls a nested procedure.</td>
</tr>
<tr>
<td></td>
<td>The parameter is passed to the nested procedure when the step runs.</td>
</tr>
<tr>
<td></td>
<td>At run time, the actual parameter name needs to match the name of a formal</td>
</tr>
<tr>
<td></td>
<td>parameter in the nested procedure.</td>
</tr>
<tr>
<td>createFormalParameter</td>
<td>Creates a new formal parameter for a procedure.</td>
</tr>
<tr>
<td>deleteActualParameter</td>
<td>Deletes an actual parameter.</td>
</tr>
<tr>
<td>deleteFormalParameter</td>
<td>Deletes a formal parameter.</td>
</tr>
<tr>
<td>detachParameter</td>
<td>Detaches a formal parameter from a step.</td>
</tr>
<tr>
<td>getActualParameter</td>
<td>Retrieves an actual parameter by its name.</td>
</tr>
<tr>
<td>getActualParameters</td>
<td>Retrieves all actual parameters from a job, job step, or step.</td>
</tr>
<tr>
<td>getFormalParameter</td>
<td>Retrieves a formal parameter by its name.</td>
</tr>
<tr>
<td>getFormalParameters</td>
<td>Retrieves all formal parameters from a procedure, schedule, or step.</td>
</tr>
<tr>
<td>modifyActualParameter</td>
<td>Modifies an existing actual parameter. An actual parameter is a name/value</td>
</tr>
<tr>
<td></td>
<td>pair that is passed to a subprocedure. This command supports renaming the</td>
</tr>
<tr>
<td></td>
<td>actual parameter and setting its value.</td>
</tr>
<tr>
<td>modifyFormalParameter</td>
<td>Modifies an existing formal parameter.</td>
</tr>
</tbody>
</table>

Plugin Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deletePlugin</td>
<td>Deletes an existing plugin object without deleting the associated project or</td>
</tr>
<tr>
<td></td>
<td>files.</td>
</tr>
<tr>
<td>getPlugin</td>
<td>Retrieves an installed plugin.</td>
</tr>
<tr>
<td>getPlugins</td>
<td>Retrieves all installed plugins.</td>
</tr>
<tr>
<td>installPlugin</td>
<td>Installs a plugin from a JAR file. Extracts the JAR contents on the server</td>
</tr>
</tbody>
</table>
### Commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifyPlugin</td>
<td>Modifies a plugin.</td>
</tr>
<tr>
<td>promotePlugin</td>
<td>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/&lt;key&gt;]</code> or a plugin name argument without a specified version.</td>
</tr>
<tr>
<td>uninstallPlugin</td>
<td>Uninstalls a plugin, deleting the associated project and any installed files.</td>
</tr>
</tbody>
</table>

### Procedure Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createProcedure</td>
<td>Creates a new procedure for an existing project.</td>
</tr>
<tr>
<td>createStep</td>
<td>Creates a new procedure step.</td>
</tr>
<tr>
<td>deleteProcedure</td>
<td>Deletes a procedure, including all steps.</td>
</tr>
<tr>
<td>deleteStep</td>
<td>Deletes a step from a procedure.</td>
</tr>
<tr>
<td>getProcedure</td>
<td>Finds a procedure by its name.</td>
</tr>
<tr>
<td>getProcedures</td>
<td>Retrieves all procedures in a project.</td>
</tr>
<tr>
<td>getStep</td>
<td>Retrieves a step from a procedure.</td>
</tr>
<tr>
<td>getSteps</td>
<td>Retrieves all steps in a procedure.</td>
</tr>
<tr>
<td>modifyProcedure</td>
<td>Modifies an existing procedure.</td>
</tr>
<tr>
<td>modifyStep</td>
<td>Modifies an existing step.</td>
</tr>
<tr>
<td>moveStep</td>
<td>Moves a step within a procedure.</td>
</tr>
</tbody>
</table>

### Project Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createProject</td>
<td>Creates a new project.</td>
</tr>
</tbody>
</table>
## Commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteProject</td>
<td>Deletes a project, including all procedures, procedure steps, and jobs.</td>
</tr>
<tr>
<td>getProject</td>
<td>Finds a project by its name.</td>
</tr>
<tr>
<td>getProjects</td>
<td>Retrieves all projects.</td>
</tr>
<tr>
<td>modifyProject</td>
<td>Modifies an existing project.</td>
</tr>
</tbody>
</table>

## Property Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createProperty</td>
<td>Creates a regular string or nested property sheet using a combination of property path and context.</td>
</tr>
<tr>
<td>deleteProperty</td>
<td>Deletes a property from a property sheet.</td>
</tr>
<tr>
<td>evalScript</td>
<td>Evaluates a script in a given context. This API is similar to <code>expandString</code> 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.</td>
</tr>
<tr>
<td>expandString</td>
<td>Expands property references in a string, in the current context.</td>
</tr>
<tr>
<td>getProperties</td>
<td>Retrieves all properties associated with an object.</td>
</tr>
<tr>
<td>getProperty</td>
<td>Retrieves the specified property value.</td>
</tr>
<tr>
<td>incrementProperty</td>
<td>Atomically increments the specified property value by the incrementBy amount. If the property does not exist, it will be created with an initial value of the incrementBy amount.</td>
</tr>
<tr>
<td>modifyProperty</td>
<td>Modifies a regular string or nested property sheet using a combination of property path and context.</td>
</tr>
<tr>
<td>setProperty</td>
<td>Sets the value for the specified property.</td>
</tr>
</tbody>
</table>

## Resource Management
### Commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addResourcesToPool</td>
<td>Adds resources to a specified resource pool.</td>
</tr>
<tr>
<td>createResource</td>
<td>Creates a new resource.</td>
</tr>
<tr>
<td>createResourcePool</td>
<td>Creates a pool container for resource.</td>
</tr>
<tr>
<td>deleteResource</td>
<td>Deletes a resource.</td>
</tr>
<tr>
<td>deleteResourcePool</td>
<td>Deletes a resource pool.</td>
</tr>
<tr>
<td>getResource</td>
<td>Retrieves a resource by its name.</td>
</tr>
<tr>
<td>getResources</td>
<td>Retrieves all resources.</td>
</tr>
<tr>
<td>getResourcesInPool</td>
<td>Retrieves a list of resources in a pool.</td>
</tr>
<tr>
<td>getResourcePool</td>
<td>Retrieves a specified resource pool by name.</td>
</tr>
<tr>
<td>getResourcePools</td>
<td>Retrieves a list of resource pools.</td>
</tr>
<tr>
<td>getRESOURCEUsage</td>
<td>Retrieves resource usage information.</td>
</tr>
<tr>
<td>modifyResource</td>
<td>Modifies an existing resource.</td>
</tr>
<tr>
<td>modifyResourcePool</td>
<td>Modifies an existing resource pool.</td>
</tr>
<tr>
<td>pingAllResources</td>
<td>Pings all resources.</td>
</tr>
<tr>
<td>pingResource</td>
<td>Pings one resources.</td>
</tr>
<tr>
<td>removeResourcesFromPool</td>
<td>Removes resources from a specified resource pool.</td>
</tr>
</tbody>
</table>

### Schedule Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createSchedule</td>
<td>Creates a new schedule.</td>
</tr>
<tr>
<td>deleteSchedule</td>
<td>Deletes a schedule.</td>
</tr>
<tr>
<td>getSchedule</td>
<td>Retrieves a schedule by its name.</td>
</tr>
<tr>
<td>getSchedules</td>
<td>Retrieves all schedules.</td>
</tr>
<tr>
<td>modifySchedule</td>
<td>Modifies an existing schedule.</td>
</tr>
</tbody>
</table>
## Server Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getVersions</td>
<td>Retrieves server version information.</td>
</tr>
<tr>
<td>shutdownServer</td>
<td>Shuts down the ElectricFlow server.</td>
</tr>
<tr>
<td>importLicenseData</td>
<td>Imports one or more licenses.</td>
</tr>
<tr>
<td>getAdminLicense</td>
<td>Retrieves the admin license, which can be used when all concurrent user licenses are in use.</td>
</tr>
<tr>
<td>getLicense</td>
<td>Retrieves information for one license.</td>
</tr>
<tr>
<td>getLicenses</td>
<td>Retrieves all license data.</td>
</tr>
<tr>
<td>getLicenseUsage</td>
<td>Retrieves the current license usage.</td>
</tr>
<tr>
<td>deleteLicense</td>
<td>Deletes a license.</td>
</tr>
<tr>
<td>getServerStatus</td>
<td>Returns the status of the server.</td>
</tr>
</tbody>
</table>

## User/Group Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>login</td>
<td>Logs into the server and saves the session ID for subsequent etool use.</td>
</tr>
<tr>
<td>logout</td>
<td>Logs out of the client session.</td>
</tr>
<tr>
<td>addUserToGroup</td>
<td>Adds ones or more specified users to a particular group.</td>
</tr>
<tr>
<td>createGroup</td>
<td>Creates a new local group of users.</td>
</tr>
<tr>
<td>createUser</td>
<td>Creates a new local user.</td>
</tr>
<tr>
<td>deleteGroup</td>
<td>Deletes a local group.</td>
</tr>
<tr>
<td>deleteUser</td>
<td>Deletes a local user.</td>
</tr>
<tr>
<td>getGroup</td>
<td>Retrieves a group by its name.</td>
</tr>
<tr>
<td>getGroups</td>
<td>Retrieves all groups.</td>
</tr>
</tbody>
</table>
### Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getUser</td>
<td>Retrieves a user by name.</td>
</tr>
<tr>
<td>getUsers</td>
<td>Retrieves all users.</td>
</tr>
<tr>
<td>modifyGroup</td>
<td>Modifies an existing group.</td>
</tr>
<tr>
<td>modifyUser</td>
<td>Modifies an existing user.</td>
</tr>
<tr>
<td>removeUsersFromGroup</td>
<td>Removes one or more users from a particular group.</td>
</tr>
</tbody>
</table>

### Workflow Definition Management

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createStateDefinition</td>
<td>Creates a new state definition for a workflow definition.</td>
</tr>
<tr>
<td>createTransitionDefinition</td>
<td>Creates a new transition definition for a workflow definition.</td>
</tr>
<tr>
<td>createWorkflowDefinition</td>
<td>Creates a new workflow definition for a project.</td>
</tr>
<tr>
<td>deleteStateDefinition</td>
<td>Deletes a state definition.</td>
</tr>
<tr>
<td>deleteTransitionDefinition</td>
<td>Deletes a transition definition.</td>
</tr>
<tr>
<td>deleteWorkflowDefinition</td>
<td>Deletes a workflow definition, including all state and transition definitions.</td>
</tr>
<tr>
<td>getStateDefinition</td>
<td>Finds a state definition by name.</td>
</tr>
<tr>
<td>getStateDefinitions</td>
<td>Retrieves all state definitions in a workflow definition.</td>
</tr>
<tr>
<td>getTransitionDefinition</td>
<td>Finds a transition definition by name.</td>
</tr>
<tr>
<td>getTransitionDefinitions</td>
<td>Retrieves all transition definitions in a workflow definition.</td>
</tr>
<tr>
<td>getWorkflowDefinition</td>
<td>Finds a workflow definition by name.</td>
</tr>
<tr>
<td>getWorkflowDefinitions</td>
<td>Retrieves all workflow definitions in a project.</td>
</tr>
<tr>
<td>modifyStateDefinition</td>
<td>Modifies an existing state definition.</td>
</tr>
<tr>
<td>modifyTransitionDefinition</td>
<td>Modifies an existing transition definition.</td>
</tr>
<tr>
<td>modifyWorkflowDefinition</td>
<td>Modifies an existing workflow definition.</td>
</tr>
</tbody>
</table>
### Commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>moveStateDefinition</code></td>
<td>Moves a state definition within a workflow definition.</td>
</tr>
<tr>
<td><code>moveTransitionDefinition</code></td>
<td>Moves a transition definition within a workflow definition.</td>
</tr>
</tbody>
</table>

## Workflow Management

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

## Workspace Management

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createWorkspace</code></td>
<td>Creates a new workspace.</td>
</tr>
<tr>
<td><code>deleteWorkspace</code></td>
<td>Deletes a workspace.</td>
</tr>
<tr>
<td><code>getWorkspace</code></td>
<td>Retrieves a workspace by name.</td>
</tr>
<tr>
<td><code>getWorkspaces</code></td>
<td>Retrieves all workspaces.</td>
</tr>
<tr>
<td><code>modifyWorkspace</code></td>
<td>Modifies an existing workspace.</td>
</tr>
</tbody>
</table>
## Miscellaneous

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>changeOwner</td>
<td>Changes the owner of an object.</td>
</tr>
<tr>
<td>clone</td>
<td>Makes a copy of an existing ElectricFlow project, procedure, step, schedule, resource, directory provider, email configuration, or email notifier.</td>
</tr>
<tr>
<td>countObjects</td>
<td>Returns the count of objects specified by the provided filter.</td>
</tr>
<tr>
<td>deleteObjects</td>
<td>This API deletes objects specified by the provided filters. Because of the complexity of specifying filter criteria, this API is not supported by ectool. However, all of its capabilities are supported through the Perl API.</td>
</tr>
<tr>
<td>export</td>
<td>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.</td>
</tr>
<tr>
<td>findObjects</td>
<td>Finds several different types of ElectricFlow objects—it is the underlying mechanism used to implement the ElectricFlow “Search” feature. Because of this command’s general nature and the complexity of specifying filter and sort criteria, it is not supported by ectool. Use the Perl API for the findObjects command.</td>
</tr>
<tr>
<td>getObjects</td>
<td>Used to retrieve the full object based on IDs returned by findObjects. All requested objects must be of the same objectType. See findObjects for a list of object types.</td>
</tr>
<tr>
<td>import</td>
<td>Imports data from an XML export file.</td>
</tr>
</tbody>
</table>

### API Commands Listed in Alphabetical Order

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.

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortAllJobs</td>
<td>Aborts all running jobs.</td>
</tr>
<tr>
<td>abortJob</td>
<td>Aborts a running job.</td>
</tr>
<tr>
<td>abortJobStep</td>
<td>Aborts any type of step—command step or subprocedure step.</td>
</tr>
<tr>
<td>Commands</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>addDependentsToArtifactVersion</td>
<td>Adds an artifact version query to an existing artifact. Dependent artifact versions are retrieved when the parent artifact version is retrieved.</td>
</tr>
<tr>
<td>addResourcesToPool</td>
<td>Adds resources to a specific resource pool.</td>
</tr>
<tr>
<td>addUsersToGroup</td>
<td>Adds ones or more specified users to a particular group.</td>
</tr>
<tr>
<td>attachCredential</td>
<td>Attaches a credential to an object.</td>
</tr>
<tr>
<td>attachParameter</td>
<td>Attaches a formal parameter to a step.</td>
</tr>
<tr>
<td>breakAclInheritance</td>
<td>Breaks ACL (access control list) inheritance at the given object.</td>
</tr>
<tr>
<td>changeOwner</td>
<td>Changes the owner of an object.</td>
</tr>
<tr>
<td>checkAccess</td>
<td>Checks ACL (access control list) permission information associated with an object (including inherited ACLs) for the current user.</td>
</tr>
<tr>
<td>cleanupArtifactCache</td>
<td>Deletes stale artifact versions from an artifact cache. A &quot;stale artifact version&quot; is one whose metadata was previously deleted from the ElectricFlow server.</td>
</tr>
<tr>
<td>cleanupRepository</td>
<td>Deletes stale artifact versions from the repository backing-store. A &quot;stale artifact version&quot; is one whose metadata was previously deleted from the ElectricFlow server.</td>
</tr>
<tr>
<td>clone</td>
<td>Makes a copy of an existing ElectricFlow project, procedure, step, schedule, resource, directory provider, email configuration, or email notifier.</td>
</tr>
<tr>
<td>completeJob</td>
<td>Completes an externally managed job.</td>
</tr>
<tr>
<td>completeJobStep</td>
<td>Completes an externally managed job step.</td>
</tr>
<tr>
<td>completeWorkflow</td>
<td>Marks a workflow as complete, which means transitions are no longer evaluated.</td>
</tr>
<tr>
<td>createAclEntry</td>
<td>Creates an ACE (access control list entry) on an object for a given principal.</td>
</tr>
<tr>
<td>createActualParameter</td>
<td>Creates a new actual parameter for a step that calls a nested procedure. The parameter is passed to the nested procedure when the step runs. At run time, the actual parameter name needs to match the name of a formal parameter in the nested procedure.</td>
</tr>
<tr>
<td>createArtifact</td>
<td>Creates a new artifact.</td>
</tr>
<tr>
<td>Commands</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>createCredential</td>
<td>Creates a new credential for a project.</td>
</tr>
<tr>
<td>createDirectoryProvider</td>
<td>Creates a new LDAP directory provider.</td>
</tr>
<tr>
<td>createEmailConfig</td>
<td>Creates a new email configuration.</td>
</tr>
<tr>
<td>createEmailNotifier</td>
<td>Creates an email notifier on an object specified by an emailNotifierSelector.</td>
</tr>
<tr>
<td>createFormalParameter</td>
<td>Creates a new formal parameter for a procedure.</td>
</tr>
<tr>
<td>createGateway</td>
<td>Creates a new gateway.</td>
</tr>
<tr>
<td>createGroup</td>
<td>Creates a new local group of users.</td>
</tr>
<tr>
<td>createJob</td>
<td>Creates an externally managed job.</td>
</tr>
<tr>
<td>createJobStep</td>
<td>Creates a job step in an existing job.</td>
</tr>
<tr>
<td>createProcedure</td>
<td>Creates a new procedure for an existing project.</td>
</tr>
<tr>
<td>createProject</td>
<td>Creates a new project.</td>
</tr>
<tr>
<td>createProperty</td>
<td>Creates a regular string or nested property sheet using a combination of property path and context.</td>
</tr>
<tr>
<td>createRepository</td>
<td>Creates a repository for one or more artifacts.</td>
</tr>
<tr>
<td>createResource</td>
<td>Creates a new resource.</td>
</tr>
<tr>
<td>createResourcePool</td>
<td>Creates a pool container for resources.</td>
</tr>
<tr>
<td>createSchedule</td>
<td>Creates a new schedule.</td>
</tr>
<tr>
<td>createStateDefinition</td>
<td>Creates a new state definition for a workflow definition.</td>
</tr>
<tr>
<td>createStep</td>
<td>Creates a new procedure step.</td>
</tr>
<tr>
<td>createTransitionDefinition</td>
<td>Creates a new transition definition for a workflow definition.</td>
</tr>
<tr>
<td>createUser</td>
<td>Creates a new local user.</td>
</tr>
<tr>
<td>createWorkflowDefinition</td>
<td>Creates a new workflow definition for a project.</td>
</tr>
<tr>
<td>createWorkspace</td>
<td>Creates a new workspace.</td>
</tr>
<tr>
<td>createZone</td>
<td>Creates a new zone.</td>
</tr>
<tr>
<td>Commands</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deleteAclEntry</td>
<td>Deletes an ACE on an object for a given principal.</td>
</tr>
<tr>
<td>deleteActualParameter</td>
<td>Deletes an actual parameter.</td>
</tr>
<tr>
<td>deleteArtifact</td>
<td>Deletes an existing artifact element and all artifact versions.</td>
</tr>
<tr>
<td>deleteArtifactVersion</td>
<td>Deletes artifact version metadata from the ElectricFlow database. (This API call does not delete or remove artifacts stored on the repository machine.)</td>
</tr>
<tr>
<td>deleteCredential</td>
<td>Deletes a credential.</td>
</tr>
<tr>
<td>deleteDirectoryProvider</td>
<td>Deletes an LDAP directory provider.</td>
</tr>
<tr>
<td>deleteEmailConfig</td>
<td>Deletes an email configuration.</td>
</tr>
<tr>
<td>deleteEmailNotifier</td>
<td>Deletes an email notifier from a property sheet container.</td>
</tr>
<tr>
<td>deleteFormalParameter</td>
<td>Deletes a formal parameter.</td>
</tr>
<tr>
<td>deleteGateway</td>
<td>Deletes a gateway.</td>
</tr>
<tr>
<td>deleteGroup</td>
<td>Deletes a local group.</td>
</tr>
<tr>
<td>deleteJob</td>
<td>Deletes a job from the ElectricFlow database.</td>
</tr>
<tr>
<td>deleteLicense</td>
<td>Deletes a license.</td>
</tr>
<tr>
<td>deleteObjects</td>
<td>This API deletes objects specified by the provided filters. Because of the complexity of specifying filter criteria, this API is not supported by ectool. However, all of its capabilities are supported through the Perl API.</td>
</tr>
<tr>
<td>deletePlugin</td>
<td>Deletes an existing plugin object without deleting the associated project or files.</td>
</tr>
<tr>
<td>deleteProcedure</td>
<td>Deletes a procedure, including all steps.</td>
</tr>
<tr>
<td>deleteProject</td>
<td>Deletes a project, including all procedures, procedure steps, and jobs.</td>
</tr>
<tr>
<td>deleteProperty</td>
<td>Deletes a property from a property sheet.</td>
</tr>
<tr>
<td>deleteRepository</td>
<td>Deletes artifact repository metadata from the ElectricFlow database. (This API call does not delete or remove artifacts stored on the repository machine.)</td>
</tr>
<tr>
<td>deleteResource</td>
<td>Deletes a resource.</td>
</tr>
<tr>
<td>Commands</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deleteResourcePool</td>
<td>Deletes a resource pool.</td>
</tr>
<tr>
<td>deleteSchedule</td>
<td>Deletes a schedule.</td>
</tr>
<tr>
<td>deleteStateDefinition</td>
<td>Deletes a state definition.</td>
</tr>
<tr>
<td>deleteStep</td>
<td>Deletes a step from a procedure.</td>
</tr>
<tr>
<td>deleteTransitionDefinition</td>
<td>Deletes a transition definition.</td>
</tr>
<tr>
<td>deleteUser</td>
<td>Deletes a local user.</td>
</tr>
<tr>
<td>deleteWorkflow</td>
<td>Deletes a workflow, including all states and transitions.</td>
</tr>
<tr>
<td>deleteWorkflowDefinition</td>
<td>Deletes a workflow definition, including all state and transition definitions.</td>
</tr>
<tr>
<td>deleteWorkspace</td>
<td>Deletes a workspace.</td>
</tr>
<tr>
<td>deleteZone</td>
<td>Deletes a zone.</td>
</tr>
<tr>
<td>detachCredential</td>
<td>Detaches a credential from an object.</td>
</tr>
<tr>
<td>detachParameter</td>
<td>Detaches a formal parameter from a step.</td>
</tr>
<tr>
<td>evalScript</td>
<td>Evaluates a script in a given context. This API is similar to expandString except that it evaluates the value argument as a Javascript block, without performing any property substitution on either the script or the result. The string value of the final expression in the script is returned as the value element of the response.</td>
</tr>
<tr>
<td>expandString</td>
<td>Expands property references in a string, in the current context.</td>
</tr>
<tr>
<td>export</td>
<td>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.</td>
</tr>
<tr>
<td>findArtifactVersions</td>
<td>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 &quot;available&quot; state, and if run in a jobstep, registers the step as a retriever for the returned artifact versions. Use the Perl API for the findArtifactVersions command.</td>
</tr>
<tr>
<td>findJobSteps</td>
<td>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.</td>
</tr>
</tbody>
</table>
## ElectricFlow Perl API Commands

<table>
<thead>
<tr>
<th>Commands</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>findObjects</strong></td>
<td>This command finds several different types of ElectricFlow objects—it is the underlying mechanism used to implement the ElectricFlow &quot;Search&quot; feature. Because of this command's general nature and the complexity of specifying filter and sort criteria, it is not supported by ectool. Use the Perl API for the findObjects command.</td>
</tr>
<tr>
<td><strong>getAccess</strong></td>
<td>Retrieves ACL information associated with an object, including inherited ACLs.</td>
</tr>
<tr>
<td><strong>getAclEntry</strong></td>
<td>Retrieves an ACE on an object for a given principal.</td>
</tr>
<tr>
<td><strong>getActualParameter</strong></td>
<td>Retrieves an actual parameter by its name.</td>
</tr>
<tr>
<td><strong>getActualParameters</strong></td>
<td>Retrieves all actual parameters from a job, jobstep, or step.</td>
</tr>
<tr>
<td><strong>getAdminLicense</strong></td>
<td>Retrieves the admin license, which can be used when all concurrent user licenses are in use.</td>
</tr>
<tr>
<td><strong>getArtifact</strong></td>
<td>Retrieves an artifact by its name.</td>
</tr>
<tr>
<td><strong>getArtifacts</strong></td>
<td>Retrieves all artifacts in the system.</td>
</tr>
<tr>
<td><strong>getArtifactVersion</strong></td>
<td>Retrieves an artifact version by its name.</td>
</tr>
<tr>
<td><strong>getArtifactVersions</strong></td>
<td>Retrieves all artifact versions in the system, filtered by artifact name, retriever job ID, and/or retriever job step ID.</td>
</tr>
<tr>
<td><strong>getCredential</strong></td>
<td>Finds a credential by name.</td>
</tr>
<tr>
<td><strong>getCredentials</strong></td>
<td>Retrieves all credentials in a project.</td>
</tr>
<tr>
<td><strong>getDatabaseConfiguration</strong></td>
<td>Retrieves the current database configuration.</td>
</tr>
<tr>
<td><strong>getDirectoryProvider</strong></td>
<td>Retrieves an LDAP directory provider by name.</td>
</tr>
<tr>
<td><strong>getDirectoryProviders</strong></td>
<td>Retrieves all LDAP directory providers.</td>
</tr>
<tr>
<td><strong>getEmailConfig</strong></td>
<td>Retrieves an email configuration by name.</td>
</tr>
<tr>
<td><strong>getEmailConfigs</strong></td>
<td>Retrieves all email configurations.</td>
</tr>
<tr>
<td><strong>getEmailNotifier</strong></td>
<td>Retrieves an email notifier from a property sheet container.</td>
</tr>
<tr>
<td><strong>getEmailNotifiers</strong></td>
<td>Retrieves all email notifiers defined for the specified property sheet container.</td>
</tr>
<tr>
<td><strong>getFormalParameter</strong></td>
<td>Retrieves a formal parameter by its name.</td>
</tr>
<tr>
<td>Commands</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getFormalParameters</td>
<td>Retrieves all formal parameters from a procedure, schedule, or step.</td>
</tr>
<tr>
<td>getFullCredential</td>
<td>Finds a credential by name, including password, from within a running step.</td>
</tr>
<tr>
<td>getGateway</td>
<td>Finds a gateway by name.</td>
</tr>
<tr>
<td>getGateways</td>
<td>Retrieves all gateways.</td>
</tr>
<tr>
<td>getGroup</td>
<td>Retrieves a group by its name.</td>
</tr>
<tr>
<td>getGroups</td>
<td>Retrieves all groups.</td>
</tr>
<tr>
<td>getJobDetails</td>
<td>Retrieves complete information about a job, including details from each job step.</td>
</tr>
<tr>
<td>getJobInfo</td>
<td>Retrieves all information about a job, except job step information.</td>
</tr>
<tr>
<td>getJobNotes</td>
<td>Retrieves the notes property sheet from a job.</td>
</tr>
<tr>
<td>getJobs</td>
<td>Retrieves summary information for a list of jobs.</td>
</tr>
<tr>
<td>getJobsForSchedule</td>
<td>Retrieves jobs started by a specific schedule.</td>
</tr>
<tr>
<td>getJobStatus</td>
<td>Retrieves the status of a job.</td>
</tr>
<tr>
<td>getJobStepDetails</td>
<td>Retrieves details for a job step.</td>
</tr>
<tr>
<td>getJobStepStatus</td>
<td>Retrieves the status of a job step.</td>
</tr>
<tr>
<td>getLicense</td>
<td>Retrieves information for one license.</td>
</tr>
<tr>
<td>getLicenses</td>
<td>Retrieves all license data.</td>
</tr>
<tr>
<td>getLicenseUsage</td>
<td>Retrieves the current license usage.</td>
</tr>
<tr>
<td>getManifest</td>
<td>Retrieves the manifest for a specified artifact version, which includes a list of files and directories in the artifact version, plus its checksum file.</td>
</tr>
<tr>
<td>getObjects</td>
<td>The getObjects command is used to retrieve the full object based on IDs returned by findObjects. All requested objects must be of the same objectType. See findObjects for a list of object types.</td>
</tr>
<tr>
<td>getPlugin</td>
<td>Retrieves an installed plugin.</td>
</tr>
<tr>
<td>getPlugins</td>
<td>Retrieves all installed plugins.</td>
</tr>
<tr>
<td>Commands</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getProcedure</td>
<td>Finds a procedure by its name.</td>
</tr>
<tr>
<td>getProcedures</td>
<td>Retrieves all procedures in a project.</td>
</tr>
<tr>
<td>getProject</td>
<td>Finds a project by its name.</td>
</tr>
<tr>
<td>getProjects</td>
<td>Retrieves all projects.</td>
</tr>
<tr>
<td>getProperties</td>
<td>Retrieves all properties associated with an object.</td>
</tr>
<tr>
<td>getProperty</td>
<td>Retrieves the specified property value.</td>
</tr>
<tr>
<td>getRepositories</td>
<td>Retrieves all artifact repository objects known to the ElectricFlow server.</td>
</tr>
<tr>
<td>getRepository</td>
<td>Retrieves an artifact repository by its name.</td>
</tr>
<tr>
<td>getResource</td>
<td>Retrieves a resource by its name.</td>
</tr>
<tr>
<td>getResources</td>
<td>Retrieves all resources.</td>
</tr>
<tr>
<td>getResourcesInPool</td>
<td>Retrieves a list of resources in a pool.</td>
</tr>
<tr>
<td>getResourcePool</td>
<td>Retrieves a specified resource pool by name.</td>
</tr>
<tr>
<td>getResourcePools</td>
<td>Retrieves a list of resource pools.</td>
</tr>
<tr>
<td>getResourceUsage</td>
<td>Retrieves resource usage information.</td>
</tr>
<tr>
<td>getSchedule</td>
<td>Retrieves a schedule by its name.</td>
</tr>
<tr>
<td>getSchedules</td>
<td>Retrieves all schedules.</td>
</tr>
<tr>
<td>getServerStatus</td>
<td>Returns the status of the server.</td>
</tr>
<tr>
<td>getState</td>
<td>Finds a state by name.</td>
</tr>
<tr>
<td>getStates</td>
<td>Retrieves all states in a workflow.</td>
</tr>
<tr>
<td>getStateDefinition</td>
<td>Finds a state definition by name.</td>
</tr>
<tr>
<td>getStateDefinitions</td>
<td>Retrieves all state definitions in a workflow definition.</td>
</tr>
<tr>
<td>getStep</td>
<td>Retrieves a step from a procedure.</td>
</tr>
<tr>
<td>getSteps</td>
<td>Retrieves all steps in a procedure.</td>
</tr>
<tr>
<td>getTransition</td>
<td>Finds a transition by name.</td>
</tr>
</tbody>
</table>
## Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getTransitions</td>
<td>Retrieves all transitions in a workflow.</td>
</tr>
<tr>
<td>getTransitionDefinition</td>
<td>Finds a transition by name.</td>
</tr>
<tr>
<td>getTransitionDefinitions</td>
<td>Retrieves all transition definitions in a workflow definition.</td>
</tr>
<tr>
<td>getUser</td>
<td>Retrieves a user by name.</td>
</tr>
<tr>
<td>getUsers</td>
<td>Retrieves all users.</td>
</tr>
<tr>
<td>getVersions</td>
<td>Retrieves server version information.</td>
</tr>
<tr>
<td>getWorkflow</td>
<td>Finds a workflow by name.</td>
</tr>
<tr>
<td>getWorkflows</td>
<td>Retrieves all workflow instances in a project.</td>
</tr>
<tr>
<td>getWorkflowDefinition</td>
<td>Finds a workflow definition by name.</td>
</tr>
<tr>
<td>getWorkflowDefinitions</td>
<td>Retrieves all workflow definitions in a project.</td>
</tr>
<tr>
<td>getWorkspace</td>
<td>Retrieves a workspace by name.</td>
</tr>
<tr>
<td>getWorkspaces</td>
<td>Retrieves all workspaces.</td>
</tr>
<tr>
<td>getZone</td>
<td>Finds a zone by name.</td>
</tr>
<tr>
<td>getZones</td>
<td>Retrieves all zones.</td>
</tr>
<tr>
<td>import</td>
<td>Imports data from an XML export file.</td>
</tr>
<tr>
<td>importLicenseData</td>
<td>Imports one or more licenses.</td>
</tr>
<tr>
<td>incrementProperty</td>
<td>Atomically increments the specified property value by the incrementBy amount. If the property does not exist, it will be created with an initial value of the incrementBy amount.</td>
</tr>
<tr>
<td>installPlugin</td>
<td>Installs a plugin from a JAR file. Extracts the JAR contents on the server and creates a project and a plugin.</td>
</tr>
<tr>
<td>login</td>
<td>Logs into the server and saves the session ID for subsequent ectool use. The userName provided determines the permissions for commands that can be run during the session.</td>
</tr>
<tr>
<td>logout</td>
<td>Logs out of the client session.</td>
</tr>
<tr>
<td>modifyAclEntry</td>
<td>Modifies an ACE on an object for a given principal.</td>
</tr>
<tr>
<td>Commands</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>modifyActualParameter</td>
<td>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.</td>
</tr>
<tr>
<td>modifyArtifact</td>
<td>Modifies an existing artifact.</td>
</tr>
<tr>
<td>modifyArtifactVersion</td>
<td>Modifies an existing artifact version.</td>
</tr>
<tr>
<td>modifyCredential</td>
<td>Modifies an existing credential.</td>
</tr>
<tr>
<td>modifyDirectoryProvider</td>
<td>Modifies an existing LDAP directory provider.</td>
</tr>
<tr>
<td>modifyEmailConfig</td>
<td>Modifies an existing email configuration.</td>
</tr>
<tr>
<td>modifyEmailNotifier</td>
<td>Modifies an email notifier in a property sheet container specified by an emailNotifierSelector.</td>
</tr>
<tr>
<td>modifyFormalParameter</td>
<td>Modifies an existing formal parameter.</td>
</tr>
<tr>
<td>modifyGateway</td>
<td>Modifies an existing gateway.</td>
</tr>
<tr>
<td>modifyGroup</td>
<td>Modifies an existing group.</td>
</tr>
<tr>
<td>modifyJob</td>
<td>Modifies the status of an externally managed job.</td>
</tr>
<tr>
<td>modifyJobStep</td>
<td>Modifies the status of an externally managed job step.</td>
</tr>
<tr>
<td>modifyPlugin</td>
<td>Modifies an existing plugin.</td>
</tr>
<tr>
<td>modifyProcedure</td>
<td>Modifies an existing procedure.</td>
</tr>
<tr>
<td>modifyProject</td>
<td>Modifies an existing project.</td>
</tr>
<tr>
<td>modifyProperty</td>
<td>Modifies a regular string or nested property sheet using a combination of property path and context.</td>
</tr>
<tr>
<td>modifyRepository</td>
<td>Modifies an existing artifact repository.</td>
</tr>
<tr>
<td>modifyResource</td>
<td>Modifies an existing resource.</td>
</tr>
<tr>
<td>modifyResourcePool</td>
<td>Modifies an existing resource pool.</td>
</tr>
<tr>
<td>modifySchedule</td>
<td>Modifies an existing schedule.</td>
</tr>
<tr>
<td>modifyStateDefinition</td>
<td>Modifies an existing state definition.</td>
</tr>
<tr>
<td>modifyStep</td>
<td>Modifies an existing step.</td>
</tr>
<tr>
<td>Commands</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>modifyTransitionDefinition</td>
<td>Modifies an existing transition definition.</td>
</tr>
<tr>
<td>modifyUser</td>
<td>Modifies an existing user.</td>
</tr>
<tr>
<td>modifyWorkflowDefinition</td>
<td>Modifies an existing workflow definition.</td>
</tr>
<tr>
<td>modifyWorkspace</td>
<td>Modifies an existing workspace.</td>
</tr>
<tr>
<td>modifyZone</td>
<td>Modifies an existing zone.</td>
</tr>
<tr>
<td>moveDirectoryProvider</td>
<td>Moves an LDAP directory provider in front of another specified provider or to the end of the list.</td>
</tr>
<tr>
<td>moveJobs</td>
<td>Moves jobs from one project to another project.</td>
</tr>
<tr>
<td>moveRepository</td>
<td>Moves an artifact repository in front of another, specified repository or to the end of the list.</td>
</tr>
<tr>
<td>moveStateDefinition</td>
<td>Moves a state definition within a workflow definition.</td>
</tr>
<tr>
<td>moveStep</td>
<td>Moves a step within a procedure.</td>
</tr>
<tr>
<td>moveTransitionDefinition</td>
<td>Moves a transition definition within a workflow definition.</td>
</tr>
<tr>
<td>pingAllResources</td>
<td>Pings all resources.</td>
</tr>
<tr>
<td>pingResource</td>
<td>Pings one resources.</td>
</tr>
<tr>
<td>promotePlugin</td>
<td>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/&lt;key&gt;]</code> or a plugin name argument without a specified version.</td>
</tr>
<tr>
<td>publishArtifactVersion</td>
<td>Publishes an artifact version to an artifact repository.</td>
</tr>
<tr>
<td>removeDependentsFromArtifactVersion</td>
<td>Removes a list of dependent artifact versions from an existing artifact version.</td>
</tr>
<tr>
<td>removeResourcesFromPool</td>
<td>Removes resources from a specified resource pool.</td>
</tr>
<tr>
<td>removeUsersFromGroup</td>
<td>Removes one or more users from a particular group.</td>
</tr>
<tr>
<td>restoreAclInheritance</td>
<td>Restores ACL inheritance at the given object.</td>
</tr>
<tr>
<td>retrieveArtifactVersions</td>
<td>Retrieves the most recent artifact version, including its dependents, from an artifact repository.</td>
</tr>
<tr>
<td>Commands</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>runProcedure</td>
<td>Creates and starts a new job using a procedure directly or specified indirectly through a schedule.</td>
</tr>
<tr>
<td>runWorkflow</td>
<td>Runs the specified workflow definition, returns the workflow name.</td>
</tr>
<tr>
<td>sendEmail</td>
<td>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.</td>
</tr>
<tr>
<td>setDatabaseConfiguration</td>
<td>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.</td>
</tr>
<tr>
<td>setJobName</td>
<td>Sets the name of a running job.</td>
</tr>
<tr>
<td>setProperty</td>
<td>Sets the value for the specified property.</td>
</tr>
<tr>
<td>shutdownServer</td>
<td>Shuts down the ElectricFlow server.</td>
</tr>
<tr>
<td>testDirectoryProvider</td>
<td>Tests an LDAP directory provider.</td>
</tr>
<tr>
<td>transitionWorkflow</td>
<td>Manually transition from the workflow active state.</td>
</tr>
<tr>
<td>uninstallPlugin</td>
<td>Uninstalls a plugin, deleting the associated project and any installed files.</td>
</tr>
</tbody>
</table>

**API Commands - ACL Management**

- `breakAclInheritance` on page 76
- `checkAccess` on page 80
- `createAclEntry` on page 85
- `deleteAclEntry` on page 90
- `getAccess` on page 95
- `getAclEntry` on page 100
- `modifyAclEntry` on page 104
- `restoreAclInheritance` on page 110
**breakAclInheritance**

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application. Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier. Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact. Argument type: String</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component. Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential that can be one of these formats:</td>
</tr>
<tr>
<td></td>
<td>- relative (for example, &quot;cred1&quot;)–The credential is assumed to be in the</td>
</tr>
<tr>
<td></td>
<td>project that contains the request target object. A qualifying project</td>
</tr>
<tr>
<td></td>
<td>name is required.</td>
</tr>
<tr>
<td></td>
<td>- absolute (for example, /projects/BuildProject/credentials/cred1&quot;)–The</td>
</tr>
<tr>
<td></td>
<td>credential can be from any specified project, regardless of the project</td>
</tr>
<tr>
<td></td>
<td>where the target object is.</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) The name of the environment template. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) The name of the environment template tier. Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier. Argument type: String</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway. Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The full name of the group. For Active Directory and LDAP, the full name if the full domain name. Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>(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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) This is an object identifier returned by findObjects and getObjects. This value is a &quot;handle&quot; only for passing to API commands. The internal structure of this value is subject to change; do not parse this value. Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) The property path. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure or a path to a procedure that includes the name. If you use this argument, you must also use projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process if the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step if the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project, which can be a path. The project name is ignored for credentials, procedures, steps, and schedules if they are specified as a path. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository used for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of a resource. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) The name of the resource template.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of a schedule, which can be a path to the schedule. If you use this argument, you must also use projectName. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the stage. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step, which can be a path to the step. If you use this argument, you must also use projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) The name of the system object. System objects names include: admin</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The full name of a user. For Active Directory or LDAP, this may be user@domain. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow. Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of a workspace. Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone. Argument type: String</td>
</tr>
</tbody>
</table>

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

**Response**
None or status OK message.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* ectool breakAclInheritance ...

*Example*

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

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version.</td>
</tr>
<tr>
<td></td>
<td>Note: An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as “groupId:artifactKey:version” and the object is searched when you specify its name one of these ways. The ElectricFlow server interprets the name form correctly.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential container of the property sheet that owns the property. Specify credentialName using one of two forms:</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• absolute (for example, “projects/BuildProject/credentials/cred1”)–The credential can be from any specified project, regardless of the project target object project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) The name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) The name of the environment template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The full name of the group. For Active Directory and LDAP, this is a full domain name.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) The object identifier returned by findObjects and getObjects. This value is a &quot;handle&quot; only for passing to API commands. The internal structure of this value is subject to change. Do not parse this value.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) Property path string.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure. It can be a path to the procedure. When using this procedure, you must also use projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) 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</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) The name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule. It can be path to the schedule. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot. It can be path to the snapshot. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step. It can be a path to the step. When using this argument, you must also enter projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) System object names include: admin</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The full name of the user. For Active Directory and LDAP, the name can be user@domain. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow. Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace. Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone. Argument type: String</td>
</tr>
</tbody>
</table>
Positional arguments
Arguments to locate the object, beginning with the top-level object locator.

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

ec-perl
syntax: $cmdr->checkAccess({},);
Example
$cmdr->checkAccess ({"projectName":"Sample Project"});

ectool
syntax: ectool checkAccess ...
Example
ectool checkAccess --projectName "Sample Project"

createAclEntry
Creates an ACE (access control list entry) on an object for a given principal.
You must specify the principalType, principalName, and locator options for the object to modify.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>principalType</td>
<td>This is either user or group. Argument type: PrincipalType</td>
</tr>
<tr>
<td>principalName</td>
<td>This is either a user or a group name. Argument type: PrincipalName</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier. Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets the name form correctly.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>changePermissionsPrivilege</td>
<td>`&lt;allow</td>
</tr>
<tr>
<td></td>
<td>Argument type: Access</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential specified in one of these formats:</td>
</tr>
<tr>
<td></td>
<td>• <strong>relative</strong> (for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the requested target object.</td>
</tr>
<tr>
<td></td>
<td>• <strong>absolute</strong> (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the project for the target object.</td>
</tr>
<tr>
<td></td>
<td>When using this argument, you must also enter <code>projectName</code>.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) The name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) The name of the environment template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>executePrivilege</td>
<td>(Optional) `&lt;allow</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of a group.</td>
</tr>
<tr>
<td>jobId</td>
<td>(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.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>modifyPrivilege</td>
<td>(Optional) `&lt;allow</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) The object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) Path to the property.</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline.</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin. It is the plugin key for a promoted plugin or the plugin key and version for an unpromoted plugin.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>readPrivilege</td>
<td>(Optional) &lt;allow</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) The name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step. When using this argument, you must also enter projectName and procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) System object names include:</td>
</tr>
<tr>
<td></td>
<td>admin</td>
</tr>
<tr>
<td></td>
<td>Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The full name of the user.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

#### Positional arguments

principalType and principalName

#### Response

None or status OK message.

#### ec-perl

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

**Example**

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

#### ectool

**syntax:**
```bash
ectool createAclEntry <principalType> <principalName> ...
```

**Example**

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

### 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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>principalType</td>
<td>A user or a group: `&lt;user</td>
</tr>
<tr>
<td></td>
<td>Argument type: PrincipalType</td>
</tr>
<tr>
<td>principalName</td>
<td>The name of the user or the group.</td>
</tr>
<tr>
<td></td>
<td>Argument type: PrincipalName</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier. Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact. Argument type: String</td>
</tr>
</tbody>
</table>
| artifactVersionName    | (Optional) The name of the artifact version. Note: An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question.  
This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name. The ElectricFlow server interprets the name form correctly. Argument type: String |
| componentName          | (Optional) The name of the component. Argument type: String                                                                                                                                              |
| configName             | (Optional) The name of the email configuration. Argument type: String                                                                                                                                   |
| credentialName         | (Optional) The name of the credential specified in one of these formats:  
- relative (for example, "cred1")—The credential is assumed to be in the project that contains the requested target object.  
- absolute (for example, "/projects/BuildProject/credentials/cred1")—The credential can be from any specified project, regardless of the project for the target object. Argument type: String |
<p>| environmentName        | (Optional) The name of the environment that must be unique among all projects. Argument type: String                                                                                                    |
| environmentTemplateName| (Optional) The name of the environment template that must be unique among all projects. Argument type: String                                                                                           |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier that must be unique among all tiers for the environment template. Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier. Argument type: String</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway. Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of a group whose ACL entry you want to delete. Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>(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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier with the ACE that you want to delete. Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) An object identifier returned by <code>findObjects</code> and <code>getObjects</code>. Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) Path to the property. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin with the ACE that you want to delete. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure with the ACE that you want to delete. When you use this argument, you must also enter projectName for the project of which this procedure is a member. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project with the ACE that you want to delete. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource with the ACE that you want to delete. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template. Argument Type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule with the ACE that you want to delete. When you use this argument, you must also enter projectName from which this schedule runs procedures. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step with the ACE that you want to delete. When using this argument, you must also enter projectName and procedureName to indicate where this step resides.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) System object names include: admin</td>
</tr>
<tr>
<td></td>
<td>Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user with the ACE that you want to delete.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace with the ACL entry that you want to delete.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
## deleteAclEntry

### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone. Argument type: String</td>
</tr>
</tbody>
</table>

#### Positional arguments

principalType, principalName

#### Response

None or a status OK message.

#### ec-perl

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

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

#### ectool

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

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

## getAccess

Retrieves ACL (access control list) 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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier. Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| artifactVersionName        | (Optional) The name of the artifact version.  
  **Note:** An artifact version name is interpreted by the server as the `artifactVersionName` attribute for the artifactVersion in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name. The ElectricFlow server interprets the name form correctly.  
  Argument type: String  |
| componentName              | (Optional) The name of the component.  
  Argument type: String  |
| configName                 | (Optional) The name of the email configuration.  
  Argument type: String  |
| credentialName             | (Optional) The name of the credential specified in one of these formats:  
  - relative (for example, "cred1")–The credential is assumed to be in the project that contains the requested target object.  
  - absolute (for example, "/projects/BuildProject/credentials/cred1")–The credential can be from any specified project, regardless of the project for the target object.  
  Argument type: String  |
| emulateRestoreInheritance  | (Optional) Whether or not to include one level of broken inheritance if it exists. This argument is used for seeing what access would look like if the lowest level of broken inheritance was restored.  
  `<Boolean flag - 0|1|true|false>` If set to `true` or `1`, this argument returns ACL information to what it would be if inheritance were restored on this object.  
  Argument type: Boolean  |
| environmentName            | (Optional) The name of the environment that must be unique among all projects.  
  Argument type: String  |
| environmentTemplateName    | (Optional) Name of the environment template.  
  Argument type: String  |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of the group.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier with the ACL.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) An object identifier returned by findObjects and getObjects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) Property path.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin with the ACL.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure with the ACL. When using this argument, you must also enter projectName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project that contains the ACL that must be unique</td>
</tr>
<tr>
<td></td>
<td>among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned</td>
</tr>
<tr>
<td></td>
<td>automatically when the property sheet is created.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release which owns the property.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository for artifact management.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource with the ACL.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool with one or more resources.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule with the ACL.</td>
</tr>
<tr>
<td></td>
<td>Also requires projectName</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of a snapshot.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step containing the ACL. When using this argument, you must also enter <code>projectName</code> Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) System objects include: admin</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user with the ACL. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow. Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace with the ACL. Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone. Argument type: String</td>
</tr>
</tbody>
</table>

**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:**

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

**Example**

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

### ectool

**syntax:**

```
ectool getAccess ...
```

**Example**

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

### 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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>principalType</td>
<td>Type of principal for this ACE: user or group.</td>
</tr>
<tr>
<td></td>
<td>Argument type: PrincipalType</td>
</tr>
<tr>
<td>principalName</td>
<td>Name of the user or group for the ACE.</td>
</tr>
<tr>
<td></td>
<td>Argument type: PrincipalName</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets the name form correctly.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component.</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration.</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential specified in one of these formats:</td>
</tr>
<tr>
<td></td>
<td>- relative (for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the requested target object.</td>
</tr>
<tr>
<td></td>
<td>- absolute (for example, &quot;projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the project for the target object.</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment that must be unique among all projects.</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) Name of the environment template.</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier.</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) Name of the environment tier.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway.</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of the group.</td>
</tr>
<tr>
<td>jobId</td>
<td>(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.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) This is an object identifier returned by findObjects and getObjects. Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) Property path. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(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</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure with the ACL. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>processoName</td>
<td>(Optional) The name of the process. Argument type: String</td>
</tr>
<tr>
<td>processoStepName</td>
<td>(Optional) The name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management. Argument type: UUID</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of a schedule. When using this argument, you must also enter projectName.</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of a snapshot. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step. When using this argument, you must also enter projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) System objects include: admin</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The full name of the user. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow. Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace. Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

principalType, principalName

**Response**

One aclEntry element.

**ec-perl**

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

  **Example**

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

**ectool**

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

  **Example**

  ectool getAclEntry user “j smith” --projectName “Sample Project”

**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.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>principalType</td>
<td>This is either user or group.</td>
</tr>
<tr>
<td></td>
<td>Argument type: PrincipalType</td>
</tr>
<tr>
<td>principalName</td>
<td>This is either a user or a group name.</td>
</tr>
<tr>
<td></td>
<td>Argument type: PrincipalName</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version.</td>
</tr>
<tr>
<td></td>
<td>An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets the name form correctly.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>changePermissionsPrivilege</td>
<td>(Optional) &lt;allow</td>
</tr>
<tr>
<td></td>
<td>Argument type: Access</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential specified in one of these formats:</td>
</tr>
<tr>
<td></td>
<td>- relative (for example, “cred1”)—The credential is assumed to be in the project that contains the requested target object.</td>
</tr>
<tr>
<td></td>
<td>- absolute (for example, “/projects/BuildProject/credentials/cred1”)—The credential can be from any specified project, regardless of the project for the target object.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>executePrivilege</td>
<td>(Optional) &lt;allow</td>
</tr>
<tr>
<td></td>
<td>Argument type: Access</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of the group with the ACL entry.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>modifyPrivilege</td>
<td>(Optional) `&lt;allow</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier with the ACL entry. Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) The object identifier returned by <code>findObjects</code> and <code>getObjects</code>. Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) Property path. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin with the ACL entry. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure with the ACL entry. When using this argument, you must also enter <code>projectName</code>. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project with the ACL entry. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>readPrivilege</td>
<td>(Optional) `&lt;allow</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource containing the ACL entry. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule with the ACL entry. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>The name of a snapshot. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step with the ACL entry. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
</tbody>
</table>
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>systemObjectName</td>
<td>(Optional) System object names include:</td>
</tr>
<tr>
<td></td>
<td>admin</td>
</tr>
<tr>
<td></td>
<td>Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user containing the ACL entry.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace containing the ACL entry.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

principalType, principalName

### Response

None or a status OK message.

### ec-perl

**Syntax:**

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

**Example:**

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

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

**Example**

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

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier. Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact. Argument type: String</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets the name form correctly. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component. Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential specified in one of these formats:</td>
</tr>
<tr>
<td></td>
<td>• relative (for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the requested target object.</td>
</tr>
<tr>
<td></td>
<td>• absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the project for the target object.</td>
</tr>
<tr>
<td></td>
<td>When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier. Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier. Argument type: String</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway. Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of the group with the ACL inheritance that you want to restore. Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| notifierName    | (Optional) The name of the email notifier with the ACL inheritance that you want to restore.  
**Also requires**  projectName and procedureName;  projectName, procedureName, and stepName;  jobId or  jobStepId  
Argument type: String |
| objectId        | (Optional) This is an object identifier returned by findObjects and getObjects. |
| path            | (Optional) Property path.  
Argument type: String |
| pipelineName    | (Optional) The name of the pipeline.  
Argument type: String |
| pluginName      | (Optional) The name of the plugin with the ACL inheritance that you want to restore.  
Argument type: String |
| procedureName   | (Optional) The name of the procedure with the ACL inheritance that you want to restore. When using this argument, you must also enter projectName.  
Argument type: String |
| processName     | (Optional) The name of the process.  
Argument type: String |
| processStepName | (Optional) The name of the process step.  
Argument type: String |
| projectName     | (Optional) The name of the project with the ACL inheritance that you want to restore.  
Argument type: String |
| propertySheetId | (Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created.  
Argument type: UUID |
| releaseName     | (Optional) The name of the release which owns the property.  
Argument type: String |
| repositoryName  | (Optional) The name of the repository for artifact management.  
Argument type: String |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource whose ACL inheritance you want to restore. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule with the ACL inheritance that you want to restore. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of a snapshot. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step with the ACL inheritance that you want to restore. When using this argument, you must also enter projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) The name of the system object whose ACL inheritance you want to restore. System objects include: admin</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user with the ACL inheritance that you want to restore. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow. Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace with the ACL inheritance that you want to restore. Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

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

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `cmdr->restoreAclInheritance({<optionals>});`

*Example*

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

**ectool**

*Syntax:* `ectool restoreAclInheritance ...`

*Example*

```
ectool restoreAclInheritance --projectName "Sample Project"
```
API Commands - Applications

- **createApplication** on page 115
- **deleteApplication** on page 116
- **getApplication** on page 116
- **getApplications** on page 117
- **modifyApplication** on page 118

**createApplication**

Creates a new application for a project.

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

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

**Positional arguments**

**projectName, applicationName**

**Response**

Returns an application element.

**ec-perl**

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

**Example**

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

**ectool**

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

**Example**

```
ectool createApplication default newApp --description aDescription
```
deleteApplication

Delete an application.

You must specify the projectName and the applicationName arguments.

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

Positional arguments

projectName, applicationName

Response

None or a status OK message.

cp-perl

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

Example

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

ectool

syntax: ectool deleteApplication <projectName> <applicationName>

Example

ectool deleteApplication default appToDelete

getApplication

Finds an application by name.

You must specify the projectName and the applicationName arguments.
### Arguments

<table>
<thead>
<tr>
<th></th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationEntityRevisionId</td>
<td>(Optional) The revision ID of the versioned object.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
</tbody>
</table>

### Positional arguments

- projectName, applicationName

### Response

 Retrieves the specified application element.

### ec-perl

**syntax:**

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

**Example**

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

### ectool

**syntax:**

```bash
ectool getApplication <projectName> <applicationName> [optionals...]
```

**Example**

```bash
ectool getApplication default newApp --applicationEntityRevisionId 4fa765dd-73f1-11e3-b67e-b0a420524153
```

### getApplications

Retrieves all applications in a project.

You must specify the `projectName` argument.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeEntityRevisions</td>
<td>(Optional) <code>&lt;Boolean flag&gt;</code> - 0</td>
</tr>
<tr>
<td>referenceComponentName</td>
<td>(Optional) Name of the master component. Argument Type: String</td>
</tr>
<tr>
<td>referenceComponentProject</td>
<td>(Optional) Project name of the master component. Argument Type: String</td>
</tr>
</tbody>
</table>

#### Positional arguments

**projectName**

#### Response

Retrieves zero or more application elements.

**ec-perl**

*Syntax:* `getApplications(<projectName>, {<optionals>});`

*Example*

```
$ec-getApplications("Default");
```

**ectool**

*Syntax:* `ectool getApplications <projectName> [optionals...]`

*Example*

```
ectool getApplications default
```

### modifyApplication

Modifies an existing application.

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>description</strong></td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td><strong>newName</strong></td>
<td>(Optional) New name for an existing object that is being renamed.</td>
</tr>
</tbody>
</table>

**Argument Type:** String

### Positional arguments

- `projectName, applicationName`

### Response

Retrieves an updated application element.

#### ec-perl

**syntax:**

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

**Example**

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

#### ectool

**syntax:**

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

**Example**

```
ectool modifyApplication default newApp --newName modApp --description exampleText
```

### API Commands - Application Tier

- `createApplicationTier` on page 119
- `deleteApplicationTier` on page 120
- `getApplicationTier` on page 121
- `getApplicationTiers` on page 122
- `getApplicationTiersInComponent` on page 123
- `modifyApplicationTier` on page 125

### createApplicationTier

Creates a new application tier in the application.

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

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

## Positional arguments

- projectName, applicationName, applicationTierName

## Response

Returns an application tier element.

### ec-perl

**syntax:**

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

**Example**

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

### ectool

**syntax:**

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

**Example**

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

## deleteApplicationTier

Deletes a tier from an application.

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

Finds an application tier by name.

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

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

#### Positional arguments

```
projectName, applicationName, applicationTierName
```

#### Response

None or a status OK message.

#### ec-perl Syntax

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

**Example**

```
$ec->deleteApplicationTier("Default", "appl", "appTierToDelete");
```

#### ectool Syntax

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

**Example**

```
ectool deleteApplicationTier default newApp appTierToDelete
```

---

### getApplicationTier

Finds an application tier by name.

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

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>Name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>Name of the tier that must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationEntityRevisionId</td>
<td>(Optional) The revision ID of the versioned object.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, applicationName, applicationTierName

**Response**

Retrieves an application tier element.

**ec-perl**

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

*Example*

$ec->getApplicationTier("Default", "app1", "appTier2", {applicationEntityRevisionId => "4fa765dd-73f1-11e3-b67e-b0a420524153");

**ectool**

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

*Example*

ectool getApplicationTier default newApp appTier1 --applicationEntityRevisionId 4fa765dd-73f1-11e3-b67e-b0a420524153

**getApplicationTiers**

Retrieves all application tiers in an application.

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>Name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationEntityRevisionId</td>
<td>(Optional) The revision ID of the versioned object.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
</tbody>
</table>

**Positional arguments**

- projectName, applicationName

**Response**

Retrieves zero or more application tier elements.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* ectool getApplicationTiers <projectName> <applicationName> [optionals...]

*Example*

ectool getApplicationTiers default newApp --applicationEntityRevisionId 4fa765dd-73f1-11e3-b67e-b0a420524153

**getApplicationTiersInComponent**

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

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

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

Back to Top
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationEntityRevisionId</td>
<td>(Optional) The revision ID of the versioned object.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) Name of an application to which this component is scoped.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>reference</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>If 1 or true, a reference of the component is created.</td>
</tr>
<tr>
<td></td>
<td>If 0 or false, a copy of the component is created.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>sourceApplicationName</td>
<td>(Optional) The name of source application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>sourceComponentName</td>
<td>(Optional) The name of new component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>sourceProjectName</td>
<td>(Optional) The name of source project.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

projectName, componentName

### Response

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

**ec-perl**

`syntax:`

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

**Example**

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

**ectool**

`syntax:`

```bash
ectool getApplicationTiersInComponent <projectName> <componentName> [optionals...]
```

**Example**

```bash
ectool getApplicationTiersInComponent default newComponent
```

Back to Top
modifyApplicationTier

Modifies an existing tier in the application.

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

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

**Positional arguments**

projectName, applicationName, applicationTierName

**Response**

Retrieves an updated application tier element.

**ec-perl**

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

**Example**

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

**ectool**

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

**Example**

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

[Back to Top]
API Commands - Artifact Management

- addDependentsToArtifactVersion on page 126
- cleanupArtifactCache on page 128
- cleanupRepository on page 128
- createArtifact on page 129
- createArtifactVersion on page 131
- createRepository on page 132
- deleteArtifact on page 133
- deleteArtifactVersion on page 134
- deleteRepository on page 134
- findArtifactVersions on page 135
- getArtifact on page 139
- getArtifacts on page 139
- getArtifactVersion on page 140
- getArtifactVersions on page 141
- getManifest on page 142
- getRepositories on page 143
- getRepository on page 144
- get RetrievedArtifacts on page 144
- modifyArtifact on page 145
- modifyArtifactVersion on page 146
- modifyRepository on page 148
- moveRepository on page 149
- publishArtifactVersion on page 150
- removeDependentsFromArtifactVersion on page 153
- resolveRoute on page 154
- retrieveArtifactVersions on page 155
- updateArtifactVersion on page 159

**addDependentsToArtifactVersion**

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

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

**Positional arguments**

<table>
<thead>
<tr>
<th>artifactVersionName</th>
<th>Response</th>
</tr>
</thead>
</table>

None or status OK message.

**ec-perl**

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

**Example**

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

**ectool**

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

**Example**

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

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

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

**Positional arguments**

`cacheDirectory`

**Response**

Returns a list of directories that were deleted.

**ec-perl**

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

*Example*

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

**ectool**

*syntax:* `ectool cleanupArtifactCache <cacheDirectory>`

*Example*

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

*Back to Top*

**cleanupRepository**

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

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

You must specify a *backingStoreDirectory*. 
Arguments | Descriptions
---|---
backingStoreDirectory | The repository directory where artifact versions are stored. Argument type: String
force | `<Boolean flag - 0|1|true|false> If set to "true", this option can be used so you can cleanup the repository even if the g/a/v s in the directory specified do not match up with any artifacts reported by the server. By default, this is false, and helps users avoid deleting arbitrary directory trees if they did not specify the repository backingstore properly. Argument type:Boolean

Positional arguments
backingStoreDirectory

Response
Returns a list of directories that were deleted.

ce-perl

```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

```bash
syntax: ectool cleanupRepository <backingStoreDirectory>

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

createArtifact

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

**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 SDK");
```

**ectool**

*Syntax:* `ectool createArtifact <groupId> <artifactKey> [optionals...]`

*Example*

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

Creates a new artifact version. 

You must specify `version`.

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

**Positional arguments**

`version`
**Response**

Returns an artifact element.

**ec-perl**

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

*Example*

$cmdr->createArtifactVersion("thirdPartyTools", "SDK", {description => "3rd party tools SDK"});

**ectool**

*Syntax:* ectool createArtifactVersion <groupId> <artifactKey> [optionals...]

*Example*

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

**createRepository**

Creates a repository for one or more artifacts.

You must specify a repositoryName.

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

### Response
Returns a `repository` element.

#### ec-perl

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

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

#### ectool

**syntax:**
```
ectool createRepository <repositoryName> [optionals...]
```

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

### deleteArtifact

Delete an existing artifact element and all artifact versions.

You must specify an `artifactName`.

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

#### 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>
```
### 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`.

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

**Argument type:** String

#### Positional arguments

- `artifactVersionName`

#### Response

None or a status OK message.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* ectool deleteArtifactVersion <artifactVersionName>

*Example*

```
ectool deleteArtifactVersion "myGroup:myKey:1.00.0-55"
```

### 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.

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

**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

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

**Two types of filters:**

Property filters are used to select objects based on the value of the object's intrinsic or custom property.

Boolean filters ("and", "or", "not") are used to combine one or more filters using Boolean logic.

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

**Property filter operators are:**

- between (2 operands)
- contains (1)
- equals (1)
- greaterOrEqual (1)
- greaterThan (1)
- in (1)
- lessOrEqual (1)
- lessThan (1)
- like (1)
- notEqual (1)
- notLike (1)
- isNotNull (0)
- isNull (0)

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

**Boolean operators are:**

- not (1 operand)
- and (2 or more operands)
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupId</td>
<td>A user-generated group name for this artifact. This field may consist of alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
</tbody>
</table>
| includeDependents  | Options are:  
- 0/false – dependent artifacts are not retrieved.  
- 1/true – dependent artifacts are retrieved. |
| jobStepId          | The unique identifier for the job step that is making the request. This job step is marked as a retriever for the matching artifact versions.          |
| versionRange       | The range of versions to search. Version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges.     |

**Positional arguments**

None

**Response**

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

**ec-perl**

**syntax:**

```perl
$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.

**getArtifact**

Retrieves an artifact by name.

You must specify an **artifactName**.

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

**Positional arguments**

artifactName

**Response**

Retrieves an **artifact** element.

**ec-perl**

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

**Example**

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

**ectool**

**syntax:** ectool getArtifact <artifactName>

**Example**

```
ectool getArtifact "myGroup:myKey"
```

**getArtifacts**

Retrieves all artifacts in the system.
You must specify search filter criteria to find the artifacts you need.

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

**Positional arguments**

None

**Response**

Zero or more `artifact` elements.

**ec-perl**

.syntax: \$cmdr->getArtifacts ()

**Example**

\$cmdr->getArtifacts ()

**ectool**

.syntax: ectool getArtifacts

**Example**

ectool getArtifacts

Back to Top

**getArtifactVersion**

Retrieves an artifact version by its name.

You must specify an `artifactVersionName`.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>includeRetrieverJobs</code></td>
<td>*(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><code>includeRetrieverJobSteps</code></td>
<td>*(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><code>maxRetrievers</code></td>
<td>*(Optional) If one of the <code>includeRetriever*</code> options are specified, return at most &quot;this many&quot; of the most recent retrievers. Without this option, the ElectricFlow server will return all retrievers. Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

`artifactVersionName`

### Response

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

#### ec-perl

**syntax:**

```perl
$cmdr->getArtifactVersion(<artifactVersionName>, {<optionals>});
```

**Example**

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

#### ectool

**syntax:**

`ectool getArtifactVersion <artifactVersionName> [optionals...]`

**Example**

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

### getArtifactVersions

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

You must specify search filter criteria to find the artifact versions you need. If you do not provide any options, all artifact versions in the system are returned.
### getArtifactVersions

Retrieves information about artifact versions. The manifest includes a list of files and directories in the artifact version and its checksum file.

You must specify the `artifactVersionName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| `artifactName`             | The name of the artifact for the versions to retrieve.  
Argument type: String     |
| `retrieverJobId`           | The job ID that retrieved an artifact.  
Argument type: String      |
| `retrieverJobStepId`       | The job step ID that retrieved an artifact.  
Argument type: UUID        |

#### Positional arguments

None

#### Response

Zero or more `artifactVersion` elements.

**ec-perl**

`syntax:`

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

**Example**

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

**ectool**

`syntax:`

```bash
ectool getArtifactVersions [optionals...]
```

**Example**

```bash
ectool getArtifactVersions --artifactName "myGroup:myKey"
```

[Back to Top](#)

### getManifest

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

You must specify the `artifactVersionName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| `artifactVersionName`      | The name of the artifact version whose manifest you want to retrieve.  
Argument type: String      |

#### Positional arguments

None
**Response**
Manifest information for the specified artifact version: returns an XML stream containing any number of file elements, including the file name, file size, and "sha1" hashes for every file in the `artifactVersionName`.

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

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

**ectool**
`syntax:` `ectool getManifest <artifactVersionName>`

**Example**
`ectool` getManifest myGroup:myKey:1.0.0-55

---

**getRepositories**
Retrieves all artifact repository objects known to the ElectricFlow server.

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

**Positional arguments**
None

**Response**
Zero or more `repository` elements.

**ec-perl**
`syntax:` `$cmdr->getRepository();`

**Example**
`$cmdr->getRepository();`

**ectool**
`syntax:` `ectoolgetRepository`

**Example**
`ectool` getRepositories

---

Back to Top
**getRepository**

Retrieves an artifact repository by its name.

You must specify a `repositoryName`.

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

**Positional arguments**

- `repositoryName`

**Response**

One `repository` element.

**ec-perl**

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

*Example*

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

**ec tool**

*Syntax:* `ectool getRepository <repositoryName>`

*Example*

`ectool getRepository myRepository`

**getRetrievedArtifacts**

Gets artifacts retrieved during job.

You must specify an `jobId`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The primary key of the job, or the name of the job.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) Name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) Name of the resource.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
**Positional arguments**

**JobId**

**Response**

Retrieved artifacts from the job.

**ec-perl**

**syntax:** `$cmdr->getRetrievedArtifacts(<jobId>, [<optionals>]);`

**Example**

```perl
$cmdr->getRetrievedArtifacts("Current components", {componentName => "WAR files"});
```

**ectool**

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

**Example**

```shell
ectool getRetrievedArtifacts "Current components" --componentName "WAR files"
```

**modifyArtifact**

Modifies an existing artifact.

You must specify an **artifactName**.

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

**Positional arguments**

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

**ec-perl**

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

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

**ectool**

*Syntax:* ectool modifyArtifact <artifactName> [optionals...]

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

**modifyArtifactVersion**

Modifies an existing artifact version.

You must specify an artifactVersionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifact version to modify. &lt;p&gt;&lt;b&gt;Note:&lt;/b&gt; An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly. Argument type: String</td>
</tr>
<tr>
<td></td>
<td><em>(Optional)</em> The state of the artifact version. &lt;p&gt;&lt;publishing</td>
</tr>
<tr>
<td>dependentArtifactVersions</td>
<td><em>(Optional)</em> 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: &lt;groupId&gt;:&lt;artifactKey&gt;:&lt;versionRange&gt; (version range is optional). &lt;p&gt;&lt;b&gt;Note:&lt;/b&gt; The absence of this argument does not clear or modify the dependent artifact version list for this artifact version. Argument type: Collection</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code> Argument type: String</td>
</tr>
<tr>
<td>newName</td>
<td>(Optional) New name for this artifact version. Argument type: String</td>
</tr>
<tr>
<td>removeAllDependentArtifactVersions</td>
<td>(Optional) <code>&lt;Boolean flag</code> - 0</td>
</tr>
<tr>
<td></td>
<td>Defaults to &quot;false.&quot; Removes all dependent artifacts from this artifact version. Subsequent &quot;retrieves&quot; will no longer retrieve dependent artifacts for this artifact version. Argument type: Boolean</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the artifact repository. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

artifactVersionName

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `cmdr->modifyArtifactVersion(<artifactVersionName>, [<optionals>]);`

*Example*

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

**ectool**

*Syntax:* `ectool modifyArtifactVersion <artifactVersionName> [optionals...]`

*Example*

```bash
ectool modifyArtifactVersion "myGroup:myKey:1.0.1-57385" --artifactVersionState unavailable
```

Back to Top
**modifyRepository**

Modifies an existing artifact repository.

You must specify a `repositoryName`.

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

**Positional arguments**

`repositoryName`

**Response**

Returns a modified `repository` element.

**ec-perl**

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

*Example*

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

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

You must specify a repositoryName.

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

**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
```
**publishArtifactVersion**

Publishes an artifact version to an artifact repository.

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactName</td>
<td>The name of an artifact. Argument type: String</td>
</tr>
<tr>
<td>artifactKey</td>
<td>User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String</td>
</tr>
<tr>
<td>compress</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>dependentArtifactVersions</td>
<td>One or more artifact version queries. The most current match of each query is retrieved when the primary artifact is retrieved. Dependent artifact version query strings are in this form: &lt;groupId&gt;:&lt;artifactKey&gt;:&lt;versionRange&gt; (versionRange is optional). The version range syntax is standard number interval notation. () marks exclusive ranges and [] marks inclusive ranges. Argument type: Collection</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt; Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>excludePatterns</td>
<td>Semi-colon delimited list of file-path patterns indicating which files/directories under &quot;fromDirectory&quot; to exclude when publishing an artifact version. Defaults to &quot;empty,&quot; which means no files are excluded. See more information on &quot;pattern syntax&quot; below. Argument type: Collection</td>
</tr>
<tr>
<td>followSymlinks</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>fromDirectory</td>
<td>The directory containing files to publish as the artifact version. A subset of files can be published based on includePatterns and excludePatterns. Argument type: String</td>
</tr>
<tr>
<td>groupId</td>
<td>A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods. Argument type: String</td>
</tr>
<tr>
<td>includePatterns</td>
<td>Semi-colon delimited list of file-path patterns indicating which files/directories under &quot;fromDirectory&quot; to publish in the artifact version. Defaults to &quot;empty,&quot; which means all files will be included. Conversely, if only two files are &quot;included,&quot; no other files except those two will be included. See more information on &quot;pattern syntax&quot; below. Argument type: Collection</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository where you want to publish. Argument type: String</td>
</tr>
<tr>
<td>version</td>
<td>Unique identifier for the artifact version in the form: major.minor.patch-qualifier-buildNumber major, minor, patch, and buildNumber are integers and qualifier can contain any character except the following: <code>\:&lt;&gt;|?*</code> If a version argument is provided, but does not follow the above format, the version will be considered 0.0.0-&lt;user-specified-version-arg&gt;-0 implicitly. See the examples below. Argument type: String</td>
</tr>
</tbody>
</table>
Version number examples

<table>
<thead>
<tr>
<th>User Input</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Major.Minor.Patch</td>
</tr>
<tr>
<td>1</td>
<td>1.0.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0.0</td>
</tr>
<tr>
<td>1.0-frank</td>
<td>1.0.0</td>
</tr>
<tr>
<td>1.0-36</td>
<td>1.0.0</td>
</tr>
<tr>
<td>1.0-frank-36</td>
<td>1.0.0</td>
</tr>
</tbody>
</table>

Pattern syntax

Include / exclude patterns are expressed as relative paths under the fromDirectory.

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

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

Examples:

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

Positional arguments

None

Response

One artifactVersion element.

cp-perl

**syntax:**

```perl
$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"

---

## removeDependentsFromArtifactVersion

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

You must specify the artifactVersionName.

### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| artifactVersionName        | The name of the artifact version from which you want to remove dependents.  
**Note:** An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name—the ElectricFlow server interprets either name form correctly.  
Argument type: String |

| dependentArtifactVersions  | (Optional) One or more artifact version queries. The most current match of each query is retrieved when the primary artifact is retrieved.  
Dependent artifact version query strings are in this form:  
<groupId>:<artifactKey>:<versionRange>  
(versionRange is optional).  
The version range syntax is standard number interval notation.  
() marks exclusive ranges and [] marks inclusive ranges.  
Argument type: Collection |

### Positional arguments

- artifactVersionName

### Response

None or status OK message.
ec-perl

**syntax:**
```perl
$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"
```

---

**resolveRoute**

Resolves the network route to an artifact repository.

You must specify the `toRepositoryName` argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>toRepositoryName</code></td>
<td>Name of the artifact repository.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>fromAgentId</code></td>
<td>(Optional) Identifier of the agent requesting the route to a destination agent or artifact repository.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Long</td>
</tr>
<tr>
<td><code>fromResourceName</code></td>
<td>(Optional) Name of the resource requesting the route to a destination agent or artifact repository.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`toRepositoryName`

**Response**

None or a status OK message.

ec-perl

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

```perl
$cmdr->resolveRoute("WebServer", {fromResourceName => "admin"});
```

ectool

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

**Example**

```bash
ectool resolveRoute "WebServer" --fromResourceName "admin"
```

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.

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

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

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

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

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

**Boolean operators are:**
- `not` (1 operand)
- `and` (2 or more operands)
- `or` (2 or more operands)
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupId</td>
<td>A user-generated group name for this artifact. This field may consist of</td>
</tr>
<tr>
<td></td>
<td>alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>includeDependents</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>• The artifact and its dependents are retrieved.</td>
</tr>
<tr>
<td></td>
<td>• The published artifact version includes the artifact's dependents, such as</td>
</tr>
<tr>
<td></td>
<td>a list of one or more artifact versions.</td>
</tr>
<tr>
<td></td>
<td>• The dependent artifact versions are stored in a subdirectory under the</td>
</tr>
<tr>
<td></td>
<td>cacheDirectory or if toDirectory is specified, under the oDirectory/ec_</td>
</tr>
<tr>
<td></td>
<td>dependent_artifacts directory.</td>
</tr>
<tr>
<td></td>
<td>If the value is 0 or false, only the artifact is retrieved. The artifact</td>
</tr>
<tr>
<td></td>
<td>version does not include the dependent artifacts.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>overwrite</td>
<td>Options are:</td>
</tr>
<tr>
<td></td>
<td>• true - deletes previous content in the directory and replaces the content</td>
</tr>
<tr>
<td></td>
<td>with your new version.</td>
</tr>
<tr>
<td></td>
<td>• false - (existing behavior) if the directory does not exist, one will be</td>
</tr>
<tr>
<td></td>
<td>created and filled with the artifact's content. If the directory exists,</td>
</tr>
<tr>
<td></td>
<td>a new directory is created with a unique name and the artifact contents</td>
</tr>
<tr>
<td></td>
<td>is supplied there.</td>
</tr>
<tr>
<td></td>
<td>• update - this is similar to a merge operation—two artifact versions can</td>
</tr>
<tr>
<td></td>
<td>be moved into the same directory, but individual files with the same</td>
</tr>
<tr>
<td></td>
<td>name will be overwritten.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
repositoryNames | A space-separated list of artifact repository names. Retrieval is attempted from each specified repository in a specified order until it succeeds or all specified repositories have rejected the retrieval. If not specified, and if this request is made in a job step context, a preferred list of repository names is obtained from the Resource definition in the server. If that list is empty, the global repository list is used. Argument type: String
retryNumber | Number of retry attempts for the operation. The default is 1. The time between retry attempts is 20 seconds. Argument type: Integer
toDirectory | Used to retrieve an artifact version to a specific directory without imposing the structure of a cache directory. Specify the full path to the new directory. - If the artifact version is in a local cache directory, it will be copied out of the cache. - If the artifact version is not in a cache directory, it will be downloaded directly to the specified directory, without putting it into a cache. toDirectory overrides cacheDirectory for downloads. Argument type: String
versionRange | The range of versions to search. Version range syntax is standard number interval notation. () marks exclusive ranges and [ ] marks inclusive ranges. Argument type: String

Positional arguments
None

Response
Returns one or more artifactVersion elements.

ee-perl

syntax:

```perl
$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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactKey</td>
<td>User-specified identifier for this artifact. This field is limited to</td>
</tr>
<tr>
<td></td>
<td>alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object.</td>
</tr>
<tr>
<td></td>
<td>If using HTML, you must surround your text with</td>
</tr>
<tr>
<td></td>
<td>&lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt;</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>excludePatterns</td>
<td>Semi-colon delimited list of file-path patterns indicating which files/directories under &quot;fromDirectory&quot; to exclude when publishing an artifact version. Defaults to &quot;empty,&quot; which means no files are excluded. See more information on &quot;pattern syntax&quot; below.</td>
</tr>
<tr>
<td>followSymlinks</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>fromDirectory</td>
<td>The directory containing files to publish as the artifact version. A subset of files can be published based on includePatterns and excludePatterns.</td>
</tr>
<tr>
<td>includePatterns</td>
<td>Semi-colon delimited list of file-path patterns indicating which files/directories under &quot;fromDirectory&quot; to publish in the artifact version. Defaults to &quot;empty,&quot; which means all files will be included. Conversely, if only two files are &quot;included,&quot; no other files except those two will be included. See more information on &quot;pattern syntax&quot; below.</td>
</tr>
<tr>
<td>newVersion</td>
<td>Unique identifier for the new artifact version in the form: major.minor.patch-qualifier-buildNumber major, minor, patch, and buildNumber are integers and qualifier can contain any character except the following: \:&lt;</td>
</tr>
<tr>
<td>path</td>
<td>The path of the original artifact under which one or more files will be added or replaced. The default path is the root.</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Unique identifier for the artifact version in the form: major.minor.patch-qualifier-buildNumber major, minor, patch, and buildNumber are integers and qualifier can contain any character except the following: `:&lt;&gt;!</td>
</tr>
</tbody>
</table>

**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"
```

### API Commands – Change History

- getDeploymentHistoryItems on page 162
- getEntityChange on page 163
- getEntityChangeDetails on page 164
- pruneChangeHistory on page 165
- revert on page 166
- searchEntityChange on page 166
getDeploymentHistoryItems

Gets all the deployment history items for a specific environment.

You must specify `projectName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The application that owns the deployment history item. Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment where the application runs. Argument type: String</td>
</tr>
<tr>
<td>latest</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The process that owns the deployment history item. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The snapshot that owns the deployment history item. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`

**Response**

Zero or more deployment history items.

**ec-perl**

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

*Example*

```perl
$cmdr->getSnapshot ("Demo Project", {applicationName => "Demo App"});
```

**ectool**

*Syntax:* `ectool getSnapshot <projectName> [optionals...]`

*Example*

```bash
ectool getSnapshot "Demo Project" --applicationName "Demo App"
```
**getEntityChange**

Retrieves entity changes.

You must specify `timeSince`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>timeSince</code></td>
<td>Start of the time interval for changes.</td>
</tr>
<tr>
<td></td>
<td>This is the time line:</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Time Line" /></td>
</tr>
<tr>
<td></td>
<td>Argument Type: Long</td>
</tr>
<tr>
<td><code>entityId</code></td>
<td>Entity ID.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>entityPath</code></td>
<td>Path to the entity.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>entityType</code></td>
<td>Type of entity.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>modifiedBy</code></td>
<td>Login ID of the user who modified the object.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>timeUntil</code></td>
<td>End of the time interval for changes.</td>
</tr>
<tr>
<td></td>
<td>If this argument is not specified, the default is Now.</td>
</tr>
<tr>
<td></td>
<td>This is the time line:</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Time Line" /></td>
</tr>
<tr>
<td></td>
<td>Argument Type: Long</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `timeSince`

**Response**

Entity changes during the time interval.
**getEntityChange**

Gets 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`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>entityId</td>
<td>The entity ID.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>entityType</td>
<td>The entity type.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>revisionNumber</td>
<td>The revision number of the entity.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
</tbody>
</table>

**Positional arguments**

`entityId`, `entityType`, `revisionNumber`

**ResponseDetails.**

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**

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

You must enter `daysToKeep`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>daysToKeep</code></td>
<td>Number of days of Change History data to keep.</td>
</tr>
<tr>
<td></td>
<td>The minimum is 7.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Long</td>
</tr>
<tr>
<td><code>forcePruneAll</code></td>
<td>(Optional) Use this argument with caution. It is used most often for testing.</td>
</tr>
<tr>
<td></td>
<td>Override the specified <code>daysToKeep</code> value and prune the entire Change History,</td>
</tr>
<tr>
<td></td>
<td>keeping nothing discardable. The <code>forcePruneAll</code> value = <code>&lt;Boolean flag&gt;</code> -0</td>
</tr>
<tr>
<td></td>
<td>Defaults to &quot;false&quot;.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
</tbody>
</table>

Positional arguments

`daysToKeep`

Response

None or a status OK message.

**ec-perl**

Syntax: `$cmdr->pruneChangeHistory (<daysToKeep>, [<optionals>]);

Example

`$cmdr->pruneChangeHistory (14);`

**ectool**

Syntax: `ectool pruneChangeHistory <daysToKeep> <optionals>

Example

`ectool pruneChangeHistory 14`

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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectId</td>
<td>Object ID</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>objectType</td>
<td>Object type</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>revisionNumber</td>
<td>Revision number of the object</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Integer</td>
</tr>
</tbody>
</table>

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

searchEntityChange

Search for entity changes.

You must enter timeSince.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeSince</td>
<td>Start of the time interval for changes. ElectricFlow searches for changes since this time. This is the time line:</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Time Line" /></td>
</tr>
<tr>
<td></td>
<td>Argument type: Long</td>
</tr>
<tr>
<td>entityId</td>
<td>Entity ID. Argument Type: String</td>
</tr>
<tr>
<td>entityPath</td>
<td>Path to the entity. Argument Type: String</td>
</tr>
<tr>
<td>entityType</td>
<td>Type of entity. Argument Type: String</td>
</tr>
<tr>
<td>modifiedBy</td>
<td>Login ID of the user who modified the object. Argument Type: String</td>
</tr>
<tr>
<td>timeUntil</td>
<td>End of the time interval for changes. ElectricFlow searches for changes up to this time. If this argument is not specified, the default is Now. This is the time line:</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Time Line" /></td>
</tr>
<tr>
<td></td>
<td>Argument Type: Long</td>
</tr>
</tbody>
</table>

**Positional arguments**

timeSince

**Response**

Entity changes during the time interval.
**ec-perl**

*Syntax:* $cmdr->searchEntityChange (<timeSince>, {<optionals>});

*Example*

$cmdr->searchEntityChange(0200, {entityType => "component"}, {timeUntil => 1600});

**ectool**

*Syntax:* ectool searchEntityChange <timeSince> {<optionals>};

*Example*

ectool searchEntityChange 0600 --entityType "component" --timeUntil 1600

---

**API Commands - Component**

- **addComponentToApplicationTier** on page 168
- **copyComponent** on page 169
- **createComponent** on page 171
- **deleteComponent** on page 172
- **getComponent** on page 173
- **getComponents**
- **getComponentsinApplicationTier** on page 175
- **modifyComponent**
- **removeComponentFromApplicationTier** on page 178

---

**addComponentToApplicationTier**

Adds the specified component to the specified application tier.

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentName</td>
<td>Name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>componentProjectName</td>
<td>(Optional) Name of the project that contains the component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

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

### Response

Returns the component and specified application tier elements.

#### ec-perl

**Syntax:**

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

**Example:**

```perl
Sec->addComponentToApplicationTier("default", "newApp", "appTier1", "component1");
```

#### ectool

**Syntax:**

```bash
ectool addComponentToApplicationTier <projectName> <applicationName> <applicationTierName> <componentName> [optionals...]
```

**Example:**

```bash
ectool addComponentToApplicationTier default newApp appTier1 VCScomponent
```

### copyComponent

Creates a new component based on an existing one.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>componentName</td>
<td>Name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>newComponentName</td>
<td>Name of the new component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) Name of the tier that must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object. It is not interpreted by ElectricFlow.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>fromApplicationName</td>
<td>(Optional) Name of source application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>toApplicationName</td>
<td>(Optional) Name of source application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**
- `projectName`, `componentName`, and `newComponentName`

**Response**
Returns the new component.

**ec-perl**
Syntax:
```
$<object>-&copyComponent(<projectName>, <componentName>, <newComponentName>, {<optionals>});
```

**Example:**
```
$ec-&copyComponent("default", "App1 WAR file", "App2 WAR file", {applicationTierName => "Web Server Config"});
```

**ectool**
Syntax:
```
ectool copyComponent <projectName> <componentName> <newComponentName> [optional s...]
```

**Example:**
```
ectool copyComponent default "App1 WAR file" "App2 WAR file" --applicationTierName "Web Server Config"
```
createComponent

Creates a new component for a project.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>componentName</code></td>
<td>Name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>actualParameters</code></td>
<td>(Optional) Parameters passed as arguments to the application component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Map</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>(Optional) Name of the application to which this component is scoped.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>credentialName</code></td>
<td>(Optional) Name of the credential to attach to this component.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>description</code></td>
<td>(Optional) Comment text describing this object. It is not interpreted by ElectricFlow.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>pluginKey</code></td>
<td>(Optional) Key of the plugin.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>pluginName</code></td>
<td>(Optional) Name of the plugin.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>pluginParameters</code></td>
<td>(Optional) List of plugin parameters.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Map</td>
</tr>
<tr>
<td><code>reference</code></td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>If 1 or true, a reference of the component is created.</td>
</tr>
<tr>
<td></td>
<td>If 0 or false, a copy of the component is created.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
</tbody>
</table>
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

Response
Returns a version-controlled component element.

**ec-perl**
Syntax:

```perl
$obj->createComponent(<projectName>, <componentName>, {<optionals>});
```

**Example:**

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

**ectool**
Syntax:

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

**Example:**

```bash
ectool createComponent default component1 --description "New agent"
```

**deleteComponent**

Deletes a component.

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentName</td>
<td>Name of the component. Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) Name of an application to which this component is scoped.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

- `projectName`, `componentName`

### Response

None or a status OK message.

### ec-perl

**Syntax:**

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

**Example:**

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

### ectool

**Syntax:**

```bash
ectool deleteComponent <projectName> <componentName> [optionals...]
```

**Example:**

```bash
ectool deleteComponent default VCScomponent
```

### GetComponent

Finds a component by name.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>Name of the component. Argument Type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
applicationEntityRevisionId | (Optional) The revision ID of the versioned object. Argument type: UUID
applicationName | (Optional) Name of an application to which this component is scoped. Argument Type: String

**Positional arguments**
- projectName, componentName

**Response**
Retrieves the specified component element.

**ec-perl**
Syntax:
```perl
$<object>-&gt;getComponent(&lt;projectName&gt;, &lt;componentName&gt;, {&lt;optionals&gt;});
```

*Example:*
```perl
Sec-&gt;getComponent("default", "component1", {applicationEntityRevisionId =&gt; "4fa765dd-73f1-11e3-b67e-b0a420524153"});
```

**ectool**
Syntax:
```bash
ectool getComponent &lt;projectName&gt; &lt;componentName&gt; [optionals...]
```

*Example:*
```bash
ectool getComponent default VCScomponent --applicationEntityRevisionId 4fa765dd-73f1-11e3-b67e-b0a420524153
```

**Back to Top**

**getComponents**
Retrieves all components in a project.
You must specify the projectName argument.

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

**Positional arguments**

projectName

**Response**

Retrieves zero or more component elements.

**ec-perl**

Syntax:

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

Example:

```perl
$ec-getComponents("default", {applicationEntityRevisionId => "4fa765dd-73f1-11e3-b67e-b0a420524153");
```

**ectool**

Syntax:

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

Example:

```bash
ectool getComponents default --applicationEntityRevisionId 4fa765dd-73f1-11e3-b67e-b0a420524153
```

**getComponentsinApplicationTier**

Returns the list of components in an application tier.

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

Arguments | Descriptions
--- | ---
projectName | Name for the project that must be unique among all projects. Argument Type: String
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>Name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>Name of the tier that must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationEntityRevisionId</td>
<td>(Optional) The revision ID of the versioned object.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>includeArtifactDetail</td>
<td>(Optional) &lt;Boolean flag&gt; 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, applicationName, applicationTierName

**Response**

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

**ec-perl**

Syntax:

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

**Example:**

```
$ec-getComponentsInApplicationTier("default", "newApp", "appTier1", {applicationEntityRevisionId => "4fa765dd-73f1-11e3-b67e-b0a420524153"});
```

**ectool**

Syntax:

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

**Example:**

```
ectool getComponentsInApplicationTier default newApp appTier1 --applicationEntityRevisionId 4fa765dd-73f1-11e3-b67e-b0a420524153
```

**modifyComponent**

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

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

**Positional arguments**

- `projectName`, `componentName`
Response
Retrieves an updated component element.

**ec-perl**
Syntax:

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

*Example:*

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

**ectool**
Syntax:

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

*Example:*

```bash
ectool modifyComponent default component1 --credentialName cred1 --newName NewName
```

Back to Top

**removeComponentFromApplicationTier**
Removes the given component from the given application tier.
You must specify the **projectName**, **applicationName**, **applicationTierName**, **and** **componentName** arguments.

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

- projectName, applicationName, applicationTierName, componentName

**Response**

None or a status OK message.

**ec-perl**

Syntax:

```perl
$<object>->removeComponentFromApplicationTier(<projectName>,
    <applicationName>, <applicationTierName>, <componentName>, [{<optionals>}]));
```

**Example:**

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

**ectool**

Syntax:

```bash
ectool removeComponentFromApplicationTier <projectName> <applicationName> <applicationTierName> <componentName> [optionals...]
```

**Example:**

```bash
ectool removeComponentFromApplicationTier default newApp appTier1 VCScomponent
```

---

**API Commands - Credential Management**

- attachCredential on page 179
- createCredential on page 181
- deleteCredential on page 183
- detachCredential on page 183
- getCredential on page 186
- getCredentials on page 187
- getFullCredential on page 187
- modifyCredential on page 188

**attachCredential**

Attaches a credential to a step or a schedule.

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

You must specify `projectName`, `credentialName`, and `locator` arguments to identify a step or a schedule.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>Name of the credential in one of these forms:</td>
</tr>
<tr>
<td></td>
<td>• <strong>relative</strong> (for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>• <strong>absolute</strong> (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application process to which the credential is attached. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component or component process to which the credential is attached. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline when a credential attached to a stage task.</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of a procedure when a credential is attached to a procedure or procedure step. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of a process when a credential is attached to a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of a process step when a credential is attached to a process step. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule for running a procedure or process in the &quot;named&quot; project when a credential is attached to the schedule. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage when a credential is attached to a stage task. Argument type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
stateDefinitionName | (Optional) The name of the workflow state definition when a credential is attached to a state definition. Argument type: String
stepName | (Optional) A step name in a procedure or process in the "named" project. Argument type: String
taskName | (Optional) The name of the task when a credential is attached to a task. Argument type: String
workflowDefinitionName | (Optional) The name of the workflow definition when a credential is attached to a state definition. Argument type: String

**Positional arguments**
- projectName, credentialName

**Response**
None or status OK message.

**ec-perl**
*Syntax:* `attachCredential("projectName", "credentialName", {...});`

**Example**
```perl
attachCredential("Test Proj", "Preflight User", {procedureName => "Run Build", stepName => "Get Sources"});
```

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

**Example**
```bash
attachCredential "Test Proj" "Preflight User" --procedureName "Run Build" --stepName "Get Sources"
```

**createCredential**
Creates a new credential for a project.

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project where the credential will be stored. The name must be unique within all projects. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>The user name for the credential. Argument type: String</td>
</tr>
<tr>
<td>password</td>
<td>The password for the credential. Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code> Argument type: String</td>
</tr>
<tr>
<td>passwordRecoveryAllowed</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

### Positional arguments

projectName, credentialName, userName, password

### Response

None or status OK message.

### ec-perl

**syntax:**

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

**Example**

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

### ectool

**syntax:**

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

**Example**

```bash
ectool createCredential "Sample Project" "Build User" "build" "abc123"
```
deleteCredential

Deletes a credential.

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

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

Positional arguments

`projectName`, `credentialName`

Response

None or a status OK message.

**ec-perl**

*Syntax:* $cmdr->deleteCredential(<projectName>, <credentialName>);

*Example*

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

**ectool**

*Syntax:* ectool deleteCredential <projectName> <credentialName>

*Example*

```ectool
detachCredential "Sample Project" "Build User"
```

Back to Top

detachCredential

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>Name of the credential in one of these forms:</td>
</tr>
<tr>
<td></td>
<td>- <strong>relative</strong> (for example, &quot;cred1&quot;)–The credential is assumed to be in</td>
</tr>
<tr>
<td></td>
<td>the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>- <strong>absolute</strong> (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)–</td>
</tr>
<tr>
<td></td>
<td>The credential can be from any specified project, regardless of the</td>
</tr>
<tr>
<td></td>
<td>target object’s project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application process with the credential that you</td>
</tr>
<tr>
<td></td>
<td>want to detach.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component or component process with the credential</td>
</tr>
<tr>
<td></td>
<td>that you want to detach.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline when a credential attached to a stage</td>
</tr>
<tr>
<td></td>
<td>task.</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure with the credential that you want to</td>
</tr>
<tr>
<td></td>
<td>detach.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process with the credential that you want to</td>
</tr>
<tr>
<td></td>
<td>detach.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step with the credential that you want</td>
</tr>
<tr>
<td></td>
<td>to detach.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule where this credential is attached.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage when a credential is attached to a stage task. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the workflow state definition when a credential is attached to a state definition. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) A step name in a procedure or process in the &quot;named&quot; project. Argument type: String</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task when a credential is attached to a task. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition when a credential is attached to a state definition. Argument type: String</td>
</tr>
</tbody>
</table>

#### Positional arguments

- **projectName**, **credentialName**

#### Response

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

- **NoSuchCredential** if the specified credential does not exist.
- **NoSuchSchedule** if the specified schedule does not exist.

#### ec-perl

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

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

#### ectool

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

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

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

**getCredential**

Finds a credential by name.

You must specify `projectName` and `credentialName`.

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

**Positional arguments**

`projectName, credentialName`

**Response**

One `credential` element.

**ec-perl**

*Syntax:* `getCredential(<projectName>, <credentialName> );`

*Example*

```perl
getCredential("SampleProject", "Build User");
```

**ectool**

*Syntax:* ` ectool getCredential <projectName> <credentialName>

*Example*

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

Retrieves all credentials in a project.
You must specify a projectName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>usableOnly</td>
<td>&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

Positional arguments

projectName

Response

Zero or more credential elements.

ec-perl

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

Example

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

ectool

syntax: ectool getCredentials <projectName> ...

Example

ectool getCredentials "Sample Project" --usableOnly 1

getFullCredential

Finds a credential by name, including password, from within a running step.
You must specify the credentialName and jobStepId.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>Name of the credential in one of these forms:</td>
</tr>
<tr>
<td></td>
<td>- relative (for example, &quot;cred1&quot;)—The credential is assumed to be in the</td>
</tr>
<tr>
<td></td>
<td>project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>- absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The</td>
</tr>
<tr>
<td></td>
<td>credential can be from any specified project, regardless of the target</td>
</tr>
<tr>
<td></td>
<td>object's project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for the job step that is used to make a project</td>
</tr>
<tr>
<td></td>
<td>association.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
</tbody>
</table>

**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*

```perl
# 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*

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

Back to Top

**modifyCredential**

Modifies an existing credential.

You must specify **projectName and credentialName**.
### Arguments & Descriptions

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>Name of the credential in one of these forms:</td>
</tr>
<tr>
<td></td>
<td>- <code>relative</code> (for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>- <code>absolute</code> (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code>.</td>
</tr>
<tr>
<td>newName</td>
<td>New name of the credential. Argument type: String</td>
</tr>
<tr>
<td>password</td>
<td>The password for the specified user name. Argument type: String</td>
</tr>
<tr>
<td>passwordRecoveryAllowed</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user containing this credential. Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

- `projectName, credentialName`

### Response

None or a status OK message.

### ec-perl Syntax

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

**Example**

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

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

**Example**

ectool modifyCredential "Sample Project" "Build User" --username build

---

**API Commands - Database Configuration**

- **getDatabaseConfiguration** on page 190
- **setDatabaseConfiguration** on page 191
- **validateDatabase** on page 193

---

**getDatabaseConfiguration**

Retrieves the current database configuration.

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

**Positional arguments**

None

**Response**

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

**ec-perl**

**syntax:** $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`

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>customDatabaseDialect</code></td>
<td>Class name of the Hibernate dialect <em>(advanced use only. The server chooses an appropriate dialect based on the databaseType).</em></td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>customDatabaseDriver</code></td>
<td>Class name of the JDBC driver <em>(advanced use only. The server chooses an appropriate driver based on the databaseType).</em></td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>customDatabaseUrl</code></td>
<td>The JDBC to use <em>(advanced use only. The server composes an appropriate URL).</em></td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>databaseName</code></td>
<td>The name of the database that you want the ElectricFlow server to use. The default is commander.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>databaseType</code></td>
<td>The type of database that you want the ElectricFlow server to use. Supported database types are: &lt;builtin</td>
</tr>
<tr>
<td></td>
<td>The default is builtin.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: DatabaseType</td>
</tr>
<tr>
<td><code>hostName</code></td>
<td>The domain name or IP address of the host server machine where the database is running.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| ignorePasskeyMismatch | `<Boolean flag - 0|1|true|false>` If true, ignore a passkey fingerprint mismatch between the current passkey file and the database configuration and discard the stored credentials.  
  
  **Note**: This action discards all saved passwords.  
  Argument Type: Boolean |
| ignoreServerMismatch  | `<Boolean flag - 0|1|true|false>` If true, ignore a host name mismatch between the current server and the database configuration where the server previously started.  
  Argument Type: Boolean |
| password              | The password required to access the database.  
  setDatabaseConfiguration does not allow a passwordless database user. Make sure the database user has a password.  
  Argument Type: String |
| port                  | The port number used to access the database. The default is the server port default.  
  Argument Type: String |
| preserveSessions      | `<Boolean flag - 0|1|true|false>` When a host name mismatch between the current server and the database configuration occurs, the default behavior is to invalidate all sessions. If this argument is set to true, all sessions are preserved and the server can reconnect to running jobs. This option is used in combination with ignoreServerMismatch.  
  Argument Type: Boolean |
| userName              | The name of the user required to access the database.  
  Argument Type: String |

**Positional arguments**  
None

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

**ec-perl**

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

*Example*  
```
$cmdr->setDatabaseConfiguration({hostName => "localhost", port => 3306});
```

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

**ectool**

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

**Example**

ectool setDatabaseConfiguration --hostName localhost --port 3306

# If the database type is set to the mysql, sqlserver, or oracle and
# you want to use the builtin database

ectool setDatabaseConfiguration --databaseType builtin --databaseName builtin

---

**validateDatabase**

Performs consistency checks on the database.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Comma-separated list of options that specify the aspects of the database to validate. Argument Type: String</td>
</tr>
</tbody>
</table>

**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

---
API Commands - Directory Provider Management

createDirectoryProvider on page 194
deleteDirectoryProvider on page 197
API Commands - Directory Provider Management on page 194
modifyDirectoryProvider on page 199
moveDirectoryProvider on page 203
testDirectoryProvider on page 204

createDirectoryProvider

Creates a new Active Directory or LDAP directory provider.
You must specify a providerName, providerType, and url.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerName</td>
<td>Name for a LDAP directory provider that must be unique.</td>
</tr>
<tr>
<td></td>
<td>This human-readable name appears in the user interface to identify users and</td>
</tr>
<tr>
<td></td>
<td>groups from this provider.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>commonGroupNameAttribute</td>
<td>The attribute in a group record that contains the common group name. If</td>
</tr>
<tr>
<td></td>
<td>specified, this name is used only when searching for groups from an external</td>
</tr>
<tr>
<td></td>
<td>provider. Use this argument if the groupNameAttribute or the uniqueGroupNameAttribute is set to distinguishedName, which is not searchable.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object.</td>
</tr>
<tr>
<td></td>
<td>If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags.</td>
</tr>
<tr>
<td></td>
<td>The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;ul&gt;</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>domainName</td>
<td>(Optional) The domain name from which Active Directory servers are</td>
</tr>
<tr>
<td></td>
<td>automatically discovered.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>emailAttribute</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>enableGroups</td>
<td>&lt;Boolean flag -0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>fullUserNameAttribute</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupBase</td>
<td>The string is prepended to the basedn to construct the directory domain name (DN) that contains group records.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupMemberAttributes</td>
<td>A comma-separated attribute name list that identifies a group member. Most LDAP configurations only specify a single value, but if there is a mixture of POSIX and LDAP style groups in the directory, multiple attributes may be required.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupMemberFilter</td>
<td>(Optional) This LDAP query is performed in the groups directory context to identify groups containing a specific user as a member. Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and groupOfNames or uniqueGroupOfNames records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: &quot;{0}&quot; is replaced with the full user record DN, and &quot;{1}&quot; is replaced with the user's account name.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupNameAttribute</td>
<td>The attribute in a group record that contains the name of the group.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupSearchFilter</td>
<td>This LDAP query is performed in the context of the groups directory to enumerate group records.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| managerDn        | The domain name (DN) of a user who has read-only access to LDAP user and group directories. If this property is not specified, the server attempts to connect as an unauthenticated user. Not all servers allow anonymous read-only access.  

**Note:** This user does not need to be an admin user with modify privileges.  
Argument type: String |
| managerPassword  | If the managerDn property is set, this password is used to authenticate the manager user.  
Argument type: String |
| providerType     | Type string for a directory provider: <ldap|activedirectory>  
Argument type: ProviderType |
| realm            | This is an identifier (string) used for LDAP directory providers so users and groups (within LDAP) can be uniquely identified in "same name" collisions across multiple directory providers. The realm is appended to the user or group name when stored in the ElectricFlow server. For example, <user>@dir (where the realm is set to "dir").  
Argument type: String |
| url              | The server URL is in the form protocol://host:port/basedn. Protocol is either ldap or ldaps (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for ldap, 636 for ldaps). The basedn is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a dc= and separated by commas.  

**Note:** Spaces in the basedn must be URL encoded (%20).  
Argument type: String |
| userBase         | This string is prepended to the basedn to construct the directory DN that contains user records.  
Argument type: String |
| userNameAttribute| The attribute in a user record that contains the user's account name.  
Argument type: String |
## Arguments and Descriptions

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>userSearchFilter</td>
<td>This LDAP query is performed in the context of the user directory to search for a user by account name. The string &quot;{0}&quot; is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.</td>
</tr>
<tr>
<td>userSearchSubtree</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>useSSL</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

### Positional arguments

- **providerName**, **providerType**, **url**

### Response

None or status OK message.

### ec-perl

**Syntax:** $cmdr->createDirectoryProvider(<providerName>, {<optionals>});

**Example**

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

### ectool

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

**Example**

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

---

### deleteDirectoryProvider

Deletes an Active Directory or LDAP directory provider.

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

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

### 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

Retrieves a directory provider by name.

You must specify a **providerName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerName</td>
<td>The name of the directory provider that must be unique. Argument Type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

**providerName**

### Response

One directoryProvider element.

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

### ec-perl

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

**Example**

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

*Syntax:* ectool getDirectoryProvider <providerName>

*Example*

ectool getDirectoryProvider AD3

**modifyDirectoryProvider**

Modifies an existing LDAP directory provider.

You must specify the `providerName`.

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

Argument Type: DirectoryType
## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The LDAP server URL is in the form protocol://host:port/basedn. Protocol is either ldap or ldaps (for secure LDAP). The port is implied by the protocol, but can be overridden if it is not at the default location (389 for ldap, 636 for ldaps). The basedn is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a dc= and separated by commas. <strong>Note:</strong> Spaces in the basedn must be URL encoded (%20). Argument Type: String</td>
</tr>
<tr>
<td>userBase</td>
<td>This string is prepended to the basedn to construct the directory DN that contains user records. Argument Type: String</td>
</tr>
<tr>
<td>userNameAttribute</td>
<td>The attribute in a user record that contains the user's account name. Argument Type: String</td>
</tr>
<tr>
<td>userSearchFilter</td>
<td>This LDAP query is performed in the context of the user directory to search for a user by account name. The string &quot;{0}&quot; is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID. Argument Type: String</td>
</tr>
<tr>
<td>userSearchSubtree</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>useSSL</td>
<td>&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

### Positional arguments

- **providerName**

### Response

None or a status OK message.

### ec-perl

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

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

dctool

**syntax:** `dctool modifyDirectoryProvider <providerName> ...`

Example

```plaintext
dctool 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`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>providerName</code></td>
<td>The name of the directory provider that must be unique. Argument Type: String</td>
</tr>
<tr>
<td><code>beforeProviderName</code></td>
<td>Moves this directory provider (<code>providerName</code>) to a place before the name specified by this option. If omitted, <code>providerName</code> is moved to the end. Argument Type: String</td>
</tr>
</tbody>
</table>

Positional arguments

`providerName`

Response

None or a status OK message.

dc-perl

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

Example

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

dctool

**syntax:** `dctool moveDirectoryProvider <providerName> ...`

Example

```plaintext
dctool moveDirectoryProvider AD3 --beforeProviderName AD2
```

Back to Top
**testDirectoryProvider**

Tests that a specific user name and password combination work with the specified directory provider settings.

You must specify **userName** and **password** (the command will prompt for the password if it is omitted).

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

**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*

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

**ectool**

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

*Example*

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

**Back to Top**

**API Commands - Dynamic Enviroment**

- addResourcePoolToEnvironmentTier on page 208
- addResourceTemplateToEnvironmentTemplateTier on page 210
- addResourceToEnvironmentTemplateTier on page 211
- createEnvironmentTemplate on page 212
- createEnvironmentTemplateTier on page 213
- createEnvironmentTemplateTierMap on page 214
- createHook on page 215
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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourcePoolName</td>
<td>Name of the resource pool that must be unique among all resource pools.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>Name of the environment that must be unique among all environments.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>Name of the environment tier that must be unique among all tiers for the environment.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**
- `resourcePoolName`, `projectName`, `environmentName`, `environmentTierName`

**Response**
- None or status OK message.

**ec-perl**

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

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

**ectool**

- **syntax:**
  ```bash
  ectool addResourcePoolToEnvironmentTier <resourcePoolName> <projectName> <environmentName> <environmentTierName>
  ```

  **Example:**
  ```bash
  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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceTemplateName</td>
<td>Name of the resource template that must be unique among all resource templates. Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template. Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String</td>
</tr>
<tr>
<td>resourceCount</td>
<td>(Optional) Number of resources to be spun from the specified resource template. Argument Type: Integer</td>
</tr>
</tbody>
</table>

Positional arguments

resourceTemplateName, projectName, environmentTemplateName, environmentTemplateTierName

Response

None or a status OK message.

ec-perl

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

Example:

$ec->addResourceTemplateToEnvironmentTemplateTier("Resource1", "default", "Production", "WebServer", {resourceCount => 4});
addResourceToEnvironmentTemplateTier

Adds a resource to the specified environment template tier.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name of the resource that must be unique among all resources.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>Name for the environment template tier that must be unique among all tiers for the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Positional arguments
resourceName, projectName, environmentTemplateName, environmentTemplateTierName

Response
None or a status OK message.

ec-perl

```perl
$cmdr->addResourceToEnvironmentTemplateTier(<resourceName>, <projectName>, <environmentTemplateName>, <environmentTemplateTierName>);
```

Example:

```perl
$ec->addResourceToEnvironmentTemplateTier("Resource1", "default", "Dev1", "Tomcat");
```
createEnvironmentTemplate

Creates an environment template.
You must specify the projectName and environmentTemplateName arguments.

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

Positional arguments
projectName, environmentTemplateName

Response
Returns a environment template object.

ec-perl

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

Example:

$ec->createEnvironmentTemplate("default", "Dev1");
ectool

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

**Example:**

`ectool createEnvironmentTemplate "default" "Dev1"`

createEnvironmentTemplateTier

Creates a tier in an environment template.

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

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

**Positional arguments**

`projectName`, `environmentTemplateName`, `environmentTemplateTierName`

**Response**

Returns an environment tier in an environment template.

**ec-perl**

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

```bash
$ectool createEnvironmentTemplateTier("default", "Dev1", "Repository");
```

tool

**syntax:**

```bash
ectool createEnvironmentTemplateTier <projectName><environmentTemplateName> <environmentTemplateTierName> [optionals]
```

**Example:**

```bash
ectool createEnvironmentTemplateTier "default" "Dev1""Repository"
```

__createEnvironmentTemplateTierMap__

Creates a environment-template tier map for an application.

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

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

`projectName, applicationName, environmentProjectName, environmentTemplateName`

**Response**

Returns a tier map for an environment template.

**ec-perl**

**syntax:**

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

**Example:**

```
$ec->createEnvironmentTemplateTierMap("default", "Undeploy", "Dev1", "Repository");
```

**ectool**

**syntax:**

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

**Example:**

```
ectool createEnvironmentTemplateTierMap "default" "Undeploy" "Dev1" "Repository"
```

**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` argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>hookName</td>
<td>Name of the hook that must be unique among all hooks in the project. Argument Type: String</td>
</tr>
</tbody>
</table>
| broadcast | (Optional) Broadcast flag  
Use this flag to broadcast the hook name in the project. The broadcast value = `<Boolean flag> -0|1|true|false>`. Defaults to true or 1.  
Argument type: Boolean |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| description                | (Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a>`, `<b>`, `<br>`, `<div>`, `<dl>`, `<font>`, `<i>`, `<li>`, `<ol>`, `<p>`, `<pre>`, `<span>`, `<style>`, `<table>`, `<tc>`, `<td>`, `<th>`, `<tr>`, `<ul>`.  
This text is not interpreted by the automation platform.  
Argument type: String                                                                 |
| hookParameters             | (Optional) Parameters that are passed to the procedure.  
Argument type: Map                                                                                                                             |
| hookType                   | (Optional) Type of the hook:  
- PRE_PROVISIONING  
- POST_PROVISIONING  
- PRECONFIGURATION  
- POSTCONFIGURATION  
- PRE_TEARDOWN  
- POST_TEARDOWN  
Argument Type: String                                                                                                                      |
| procedureName              | (Optional) Name of the procedure that the hook references.  
Argument Type: String                                                                                                                        |
| procedurePluginKey         | (Optional) Name of the plugin procedure. Use this argument when the hook references a plugin procedure. The promoted version of the plugin is invoked by default.  
Argument Type: String                                                                                                                        |
| procedureProjectName       | (Optional) Name of the project to which the procedure belongs. When you use a specific version of a plugin, this is the name of the plugin project.  
Argument Type: String                                                                                                                        |
| projectName                | (Optional) Project name of the entity that owns the hook.  
Argument Type: String                                                                                                                        |
| resourceTemplateName      | (Optional) Name of the resource template.  
Argument Type: String                                                                                                                        |

**Positional arguments**

hookName
Response

Returns a hook for a resource template.

ect-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"

createResourceTemplate

Creates a resource template.

You must specify the projectName and resourceTemplateName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>Name for the resource template that must be unique among all resource templates.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>cfgMgrParameters</td>
<td>(Optional) Configuration Manager plugin parameters that are passed from the configuration-manager plugin to ElectricFlow.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Map</td>
</tr>
<tr>
<td>cfgMgrPluginKey</td>
<td>(Optional) Configuration Manager plugin key.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>cfgMgrProcedure</td>
<td>(Optional) Name of the cloud-provider plugin method.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>
### Arguments

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

### Positional arguments

- `resourceTemplateName`, `cloudProviderProcedure`, `cloudProviderConfig`

### Response

Returns a resource template.

### ec-perl

**syntax:**

```perl
$cmdr->createResourceTemplate(<projectName>, <resourceTemplateName>, {<optionals>});
```

**Example:**

```perl
$ec->createResourceTemplate("default", "QA test", {cloudProviderProjectName => "Deploy2"});
```
**ectool**

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

**Example:**

`ectool createResourceTemplate "default" "QA test" --cloudProviderProjectName "Deploy2"

**deleteEnvironmentTemplate**

Deletes an environment template.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>environmentTemplateName</code></td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `environmentTemplateName`

**Response**

None or a status OK message.

**ec-perl**

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

**Example:**

`$cmdr->deleteEnvironmentTemplate("default", "Dev1");`

**ectool**

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

**Example:**

`ectool deleteEnvironmentTemplate "default" "Dev1"`
deleteEnvironmentTemplateTier

Deletes a tier in an environment template.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>Name of the environment template tier that must be unique among all tiers for the environment template.</td>
</tr>
</tbody>
</table>

Positional arguments

projectName, environmentTemplateName, environmentTemplateTierName

Response

None or a status OK message.

del

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

Example:

$cmdr->deleteEnvironmentTemplateTier("default", "Dev1", "Repository");

del

syntax: ectool deleteEnvironmentTemplateTier <projectName>
<environmentTemplateName> <environmentTemplateTierName> [optionals]

Example:

cctool deleteEnvironmentTemplateTier "default" "Dev1" "Repository"

Back to Top

deleteEnvironmentTemplateTierMap

Deletes an environment-template tier map from an application.

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

Deletes a tier mapping from an 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Argument Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
<td>String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application.</td>
<td>String</td>
</tr>
<tr>
<td>environmentProjectName</td>
<td>Name for the project to which the environment belongs.</td>
<td>String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template.</td>
<td>String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>Name of the application tier.</td>
<td>String</td>
</tr>
</tbody>
</table>

**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", "Dev1", "Repository", "Database");
```

**ectool**

**Syntax:**
```
ectool deleteEnvironmentTemplateTierMapping <projectName> <applicationName> <environmentProjectName> <environmentTemplateName> <applicationTierName>
```

**Example:**
```
ectool deleteEnvironmentTemplateTierMapping "default" "Undeploy" "Dev1" "Repository" "Database"
```

### deleteHook

Deletes a hook associated with an entity.
You must specify the **hookName** argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>hookName</td>
<td>Name of the hook that must be unique among all hooks in the project.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) Name of the project that owns the hook.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

- **hookName**

### Response

None or a status OK message.

**ec-perl**

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

*Example:*

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

**ectool**

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

*Example:*

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

### deleteResourceTemplateName

Deletes a resource template.

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

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceTemplateName</td>
<td>Name for the resource template that must be unique among all resource templates.</td>
</tr>
</tbody>
</table>

**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"

---

### getAvailableResourcesForEnvironment

Retrieve all non-dynamic resources or resource pools.

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>includePoolUsage</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td>objectTypeToReturn</td>
<td>(Optional) Flag to return resources or resource pools.</td>
</tr>
</tbody>
</table>

**Response**

None or a status OK message.

**ec-perl**

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

GETS AN ENVIRONMENT TEMPLATE.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template. Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, environmentTemplateName

**Response**

A **environmentTemplate element.**

**ec-perl**

**syntax:** $cmdr->getEnvironmentTemplate(<projectName>, <environmentTemplateName>);

**Example:**

$cmdr->getEnvironmentTemplate("default", "Dev1");

**ectool**

**syntax:** ectool getEnvironmentTemplate <projectName> <environmentTemplateName> ...

**Example:**

ectool getEnvironmentTemplate "default" "Dev1"
getEnvironmentTemplateTier

Gets an environment tier in an environment template.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>Name of the environment template tier that must be unique among all tiers for the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Positional arguments

projectName, environmentTemplateName, environmentTemplateTierName

Response

An environmentTemplateTier element.

ec-perl

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

Example:

$ec->getEnvironmentTemplateTier("default", "Dev1", "Repository");

ectool

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

Example:

ectool createEnvironmentTemplateTier "default" "Dev1" "Repository"

Back to Top

getEnvironmentTemplateTierMaps

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

You must specify the projectName and applicationName arguments.
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>Name of the application. Argument Type: String</td>
</tr>
<tr>
<td><code>applicationEntityRevisionId</code></td>
<td>(Optional) Revision ID of the versioned object. Argument type: UUID</td>
</tr>
<tr>
<td><code>orderByEnvironmentTemplateUsage</code></td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

### Positional arguments

- `projectName`, `applicationName`

### Response

One or more `environmentTemplateTierMap` elements.

**ec-perl**

**syntax:**

```
$cmdr->getEnvironmentTemplateTierMaps(<projectName>, <applicationName>,
{<optionals>});
```

**Example:**

```
Sec-getEnvironmentTemplateTierMaps("default", "Undeploy");
```

**ectool**

**syntax:**

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

**Example:**

```
ectool getEnvironmentTemplateTierMaps "default" "Undeploy"
```

### getEnvironmentTemplateTiers

Gets all the environment template tiers in the specified environment template. You must specify the `projectName` and `environmentTemplateName` arguments.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>includeTemplateDetails</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>If this is set to &quot;true&quot;, the response includes the template details.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Boolean</td>
</tr>
</tbody>
</table>

**Positional arguments**
- projectName, environmentTemplateName

**Response**
One or more environmentTemplateTier elements.

**ec-perl**

```perl
$cmdr->getEnvironmentTemplateTiers(<projectName>, <environmentTemplateName> {<optionals>});
```

*Example:*

```
Sec->getEnvironmentTemplateTiers("default", "Dev1");
```

**ectool**

```bash
ectool getEnvironmentTemplateTiers <projectName> <environmentTemplateName> [optionals]
```

*Example:*

```
ectool getEnvironmentTemplateTiers "default" "Dev1"
```

**getEnvironmentTemplates**

Gets all the environment templates in the specified project.

You must specify the projectName argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>
Positional arguments
- projectName

Response
- One or more environmentTemplate elements.

ec-perl

**syntax:** $cmdr->getEnvironmentTemplates(<projectName>);

**Example:**
```perl
$ec->getEnvironmentTemplates("default");
```

ectool

**syntax:** ectool getEnvironmentTemplates <projectName>

**Example:**
```bash
ectool getEnvironmentTemplates "default"
```

getHook

Gets a hook associated in an entity.

You must specify the **hookName** argument.

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

Positional arguments
- hookName

Response
- Returns a hook for a resource template.

ec-perl

**syntax:** $cmdr->getHook(<hookName>, {<optionals>});
Example:

```
$ec->getHook("config", [resourceTemplateName => "Servers"]);
```

dctool

**syntax:** dctool getHook <hookName> [optionals]

**Example:**

```
dctool getHook "config" -- resourceTemplateName "Servers"
```

Back to Top

getHooks

Gets all the hook associated with an entity.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>(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</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template with the hook. Argument Type: String</td>
</tr>
</tbody>
</table>

Positional arguments

None

Response

Returns a hook for a resource template.

ec-perl

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

**Example:**

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

dctool

**syntax:** dctool getHooks [optionals]

**Example:**

```
dctool getHooks --projectName "default" --resourceTemplateName "AWS servers"
```

Back to Top
**getResourcePoolsInEnvironmentTier**

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>Name of the environment that must be unique among all environments.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>Name of the environment tier that must be unique among all tiers for the environment.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `environmentName`, `environmentTierName`

**Response**

One or more `resourcePool` elements.

**ec-perl**

`syntax:`

```perl
$cmdr->getResourcePoolsInEnvironmentTier(<projectName>, <environmentName>, <environmentTierName>);
```

**Example**

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

**ectool**

`syntax:`

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

**Example**

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

**getResourceTemplate**

Retrieves the specified resource template.

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

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

### Positional arguments

`projectName, resourceTemplateName`

### Response

A `resourceTemplate` element.

#### ec-perl

`syntax:`

```perl
$cmdr->getResourceTemplate(<projectName>, <resourceTemplateName>);
```

**Example:**

```perl
$cmdr->getResourceTemplate("default", "System Test");
```

#### ectool

`syntax:`

```bash
ectool getResourceTemplate <projectName> <resourceTemplateName>
```

**Example:**

```bash
ectool getResourceTemplate "default" "System Test"
```

### Back to Top

---

## getResourceTemplates

Retrieves all the resource templates.

You must enter the `projectName`.

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

### Positional arguments

`projectName`

### Response

One or more `resourceTemplate` elements.
**getResourcesTemplatesInEnvironmentTemplateTier**

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projectName</strong></td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><strong>environmentTemplateName</strong></td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><strong>environmentTemplateTierName</strong></td>
<td>Name of the environment template tier that must be unique among all tiers for the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

**projectName**, **environmentTemplateName**, **environmentTemplateTierName**

**Response**

One or more resourceTemplate elements.

**ec-perl**

.syntax: $cmdr->getResourceTemplatesInEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <environmentTemplateTierName>);
**Example:**

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

### ectool

**syntax:**

```bash
generateResourceTemplatesInEnvironmentTemplateTier <projectName> <environmentTemplateName> <environmentTemplateTierName>
```

**Example:**

```bash
ectool getResourceTemplatesInEnvironmentTemplateTier "default" "Production" "Web Server"
```

### getResourcesInEnvironmentTemplateTier

**Description:**

Gets all the resources in the specified environment template tier.

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

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>environmentTemplateName</code></td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>environmentTemplateTierName</code></td>
<td>Name for the environment template tier that must be unique among all tiers for the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `environmentTemplateName`, `environmentTemplateTierName`

**Response**

One or more resource elements.

### ec-perl

**syntax:**

```perl
Scmdr->getResourcesInEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <environmentTemplateTierName>);
```

**Example:**

```perl
$ec->getResourcesInEnvironmentTemplateTier("default", "Dev1", "Tomcat");
```
**ectool**

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

**Example:**
```
ectool getResourcesInEnvironmentTemplateTier "default" "Dev1" "Tomcat"
```

**modifyEnvTempTierResourceTempMapping**

Modifies the resource count in an environment template tier.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceCount</td>
<td>Number of resources to provision from the specified resource template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Integer</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>Name of the resource template that must be unique among all resource templates.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>Name for the environment template tier that must be unique among all tiers for the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`resourceCount, projectName, environmentTemplateName, resourceTemplateName, environmentTemplateTierName`

**Response**

None or a status OK message.

**ec-perl**

**syntax:**
```
$cmdr->modifyEnvTempTierResourceTempMapping(<resourceCount>, <projectName>, <environmentTemplateName>, <resourceTemplateName>, <environmentTemplateTierName>);
```
modifyEnvironmentTemplate

Creates an environment template.

You must specify the projectName and environmentTemplateName arguments.

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

Positional arguments

projectName, environmentTemplateName

Response

None or a status OK message.
**modifyEnvironmentTemplateTier**

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

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

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

- `projectName`
- `environmentTemplateName`
- `environmentTemplateTierName`

Response

None or a status OK message.

**ec-perl**

```perl
$cmdr->modifyEnvironmentTemplateTier(<projectName>, <environmentTemplateName>, <environmentTemplateTierName>, [<optionals>]);
```

*Example:*

```
$ec->modifyEnvironmentTemplateTier("default", "Dev1", "Database");
```

**ectool**

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

*Example:*

```
ectool modifyEnvironmentTemplateTier "default" "Dev1" "Database"
```

modifyEnvironmentTemplateTierMap

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>Name of the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>environmentProjectName</code></td>
<td>Name for the project to which the environment belongs.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>environmentTemplateName</code></td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>applicationEntityRevisionId</code></td>
<td>(Optional) Revision ID of the versioned object.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: UUID</td>
</tr>
</tbody>
</table>
modifyEnvironmentTemplateTierMap

Modifies an existing tier map associated with an environment template.

**Arguments**

- **tierMapName** *(Optional)* Name of the tier map associated with the environment template. If you do not specify a tier map, ElectricFlow uses a tier map with an 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**

None or a status OK message.

**ec-perl**

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

  **Example**:

  $ec-modifyEnvironmentTemplateTierMap("default", "Undeploy", "Beta", "Servers");

**ectool**

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

  **Example**:

  ectool modifyEnvironmentTemplateTierMap "default" "Undeploy" "Beta" "Servers"

modifyHook

Modifies an existing hook in a resource template.

You must specify the **hookName** argument.

**Arguments**

- **hookName** Name of the hook that must be unique among all hooks in the project.
  
  Argument Type: String
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>broadcast</td>
<td><em>(Optional)</em> Broadcast flag</td>
</tr>
</tbody>
</table>
|                           | Use this flag to broadcast the hook name in the project. The broadcast value = `<Boolean flag -0|1|true|false>`. Defaults to `true` or `1`.
|                           | Argument type: *Boolean*                                                                                                                     |
| description               | *(Optional)* A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> `<b> `<br> `<div> `<dl> `<font> `<i> `<li> `<ol> `<p> `<pre> `<span> `<style> `<table> `<tc> `<td> `<th> `<tr> `<ul>.
|                           | This text is not interpreted by the automation platform. Argument type: *String*                                                             |
| hookParameters            | *(Optional)* Parameters that are passed to the procedure. Argument type: *Map*                                                           |
| hookType                  | *(Optional)* Type of the hook:                                                                                                               |
|                           |   - PRE_PROVISIONING                                                                                                                        |
|                           |   - POST_PROVISIONING                                                                                                                       |
|                           |   - PRE_CONFIGUATION                                                                                                                        |
|                           |   - POST_CONFIGUATION                                                                                                                        |
|                           |   - PRE_TEARDOWN                                                                                                                             |
|                           |   - POST_TEARDOWN                                                                                                                            |
|                           | Argument Type: *String*                                                                                                                      |
| newName                   | New name for the hook.                                                                                                                      |
| procedureName             | Name of the procedure that the hook references. Argument Type: *String*                                                                  |
| procedurePluginKey        | *(Optional)* Name of the plugin procedure. Use this argument when the hook references a plugin procedure. The promoted version of the plugin is invoked by default. Argument Type: *String* |
| procedureProjectName      | *(Optional)* Name of the project to which the procedure belongs. When you use a specific version of a plugin, this is the name of the plugin project. Argument Type: *String* |
modifyHook

Modifies the specified resource hook.

Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>(Optional) Project name of the entity that owns the hook.</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template with the hook.</td>
</tr>
</tbody>
</table>

Argument Type: String

Positional arguments

hookName

Response

None or a status OK message.

ec-perl syntax:

```
$cmdr->modifyHook(<hookName>, {<optionals>});
```

Example:

```
$cmdr->modifyHook("config", {newName => "prod_config"});
```

ectool syntax:

```
ectool modifyHook <hookName> [optionals]
```

Example:

```
ectool modifyHook "config" --newName "prod_config"
```

modifyResourceTemplate

Modifies the specified resource template.

You must specify the projectName and resourceTemplateName arguments.

Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>Name for the resource template that must be unique among all resource templates.</td>
</tr>
</tbody>
</table>

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

**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"

provisionEnvironment
Provisions an environment.
You must specify the projectName, environmentName, and environmentTemplateName arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>Name of the environment.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name for the environment template that must be unique among all environment templates.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>keepOnError</td>
<td>(Optional) Keep on error flag</td>
</tr>
<tr>
<td></td>
<td>Set this flag to &quot;true&quot; to keep the environment when an error occurs</td>
</tr>
<tr>
<td></td>
<td>The keepOnError value = &lt;Boolean flag&gt; -0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>tierResourceCounts</td>
<td>(Optional) Resource count for each environment template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Map</td>
</tr>
</tbody>
</table>
**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**

```perl
$cmdr->provisionEnvironment(<projectName>, <environmentName>, <environmentTemplateName>, {...optionals});
```

**Example:**

```
$ec->provisionEnvironment("default", "Dev_GroupA", "BuildServer");
```

**ectool**

```bash
ectool provisionEnvironment <projectName> <environmentName> <environmentTemplateName> [...optionals]
```

**Example:**

```
ectool provisionEnvironment "default" "Dev_GroupA" "BuildServer"
```

---

**provisionResourcePool**

Provisions a resource pool.

You must specify the `resourceCount`, `resourcePoolName`, `projectName`, and `resourceTemplateName` arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resourceCount</code></td>
<td>Number of resources to provision.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>resourcePoolName</code></td>
<td>Name of the resource pool.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>projectName</code></td>
<td>Name of the project that must be unique among all projects</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>resourceTemplateName</code></td>
<td>Name for the resource template that must be unique among all resource templates.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
keepOnError | (Optional) Keep on error flag
  
  Set this flag to true or 1 to keep the environment when an error occurs.
  
  The keepOnError value = `<Boolean flag -0|1|true|false>`. Defaults to "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:**

```perl
$ec->provisionResourcePool("12", "QE_build", "default", "Servers")
```

#### ectool

**Syntax:**

```
ectool provisionResourcePool <resourceCount> <resourcePoolName> <projectName> <resourceTemplateName> [optionals]
```

**Example:**

```
ectool provisionResourcePool "12" "QE_build" "default" "Servers"
```

### removeResourceFromEnvironmentTemplateTier

Removes a resource from the specified environment template tier.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name of the resource that must be unique among all resources.</td>
</tr>
<tr>
<td>Argument Type: String</td>
<td></td>
</tr>
</tbody>
</table>
### arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>Name for the environment template tier that must be unique among all tiers for the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

#### Positional arguments

resourceName, projectName, environmentTemplateName, environmentTemplateTierName

#### Response

None or a status OK message.

#### ec-perl

**syntax:**

```
$cmdr->removeResourceFromEnvironmentTemplateTier(<resourceName>, <projectName>, <environmentTemplateName>, <environmentTemplateTierName>);
```

**Example:**

```
$ec->removeResourceFromEnvironmentTemplateTier("Resource1", "default", "Dev1", "Tomcat");
```

#### ectool

**syntax:**

```
ectool removeResourceFromEnvironmentTemplateTier <resourceName> <projectName> <environmentTemplateName> <environmentTemplateTierName>
```

**Example:**

```
ectool removeResourceFromEnvironmentTemplateTier "Resource1" "default" "Dev1" "Tomcat"
```

Back to Top

---

**removeResourcePoolFromEnvironmentTier**

Removes a resource pool from the specified environment tier.

You must specify the resourcePoolName, projectName, environmentName, and environmentTierName arguments.
Arguments | Descriptions
--- | ---
resourcePoolName | Name of the resource pool that must be unique among all resource pools. Argument Type: String
projectName | Name of the project that must be unique among all projects. Argument Type: String
environmentName | Name of the environment that must be unique among all environments. Argument Type: String
environmentTierName | Name of the environment tier that must be unique among all tiers for the environment. Argument Type: String

**Positional arguments**
resourcePoolName, projectName, environmentName, environmentTierName

**Response**
None or status OK message.

**ec-perl**

Syntax:
```
$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"
```

**removeResourceTemplateFromEnvironmentTemplateTier**

Removes a resource template from the specified environment template tier.
You must specify the `resourceTemplateName`, `projectName`, `environmentTemplateName`, and `environmentTemplateTierName` arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>resourceTemplateName</code></td>
<td>Name of the resource template that must be unique among all resource templates. Argument Type: String</td>
</tr>
<tr>
<td><code>projectName</code></td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td><code>environmentTemplateName</code></td>
<td>Name of the environment template. Argument Type: String</td>
</tr>
<tr>
<td><code>environmentTemplateTierName</code></td>
<td>Name of the environment template tier that must be unique among all tiers for the environment template. Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

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

**Response**

None or a status OK message.

**ec-perl**

syntax:

```perl
$cmdr->removeResourceTemplateFromEnvironmentTemplateTier(<resourceTemplateName>, <projectName>, <environmentTemplateName>, <environmentTemplateTierName>);
```

**Example:**

```perl
Sec->removeResourceTemplateFromEnvironmentTemplateTier("Resource1", "default", "Production", "WebServer");
```

**ectool**

**syntax:** removeResourceTemplateFromEnvironmentTemplateTier <resourceTemplateName> <projectName> <environmentTemplateName> <environmentTemplateTierName>

**Example:**

```bash
ectool removeResourceTemplateFromEnvironmentTemplateTier"Resource1" "default" "Production" "WebServer"
```
tearDown

Removes dynamic environments that are no longer needed.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentName</td>
<td>(Optional) Name of the environment. Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) Name of the resource. Argument Type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) Name of the resource pool. Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

None or a status OK message.

**ec-perl**

To tear down an environment:

**syntax:**

```bash
$cmdr->tearDown {{<optionals>}};
```

**Example:**

```bash
$ec->tearDown {{environmentName => "Server backup", projectName => "Default"}};
```

**ectool**

To tear down an environment:

**syntax:**

```bash
ectool tearDown [optionals]
```

**Example:**

```bash
ectool tearDown --environmentName "Server backup" --projectName "Default"
```

Back to Top

**API Commands - Email Configuration Management**

- [createEmailConfig](page 250) on page 250
- [deleteEmailConfig](page 251) on page 251
**createEmailConfig**

Creates a new email configuration.

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

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

* configName

Response

None or status OK message.

**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`.

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

Positional arguments

* configName

Response

None or a status OK message.

**ec-perl**

*Syntax:* `$cmdr->deleteEmailConfig(<configName>);`
**Example**

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

**ectool**

**syntax:** ectool deleteEmailConfig <configName>

**Example**

ectool deleteEmailConfig emailA

**getEmailConfig**

Retrieves an email configuration by name.

You must specify a `configName`.

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

**Positional arguments**

`configName`

**Response**

Returns one `emailConfig` element.

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

**ec-perl**

**syntax:**

```perl
$cmdr->getEmailConfig(<configName>);
```

**Example**

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

**ectool**

**syntax:** ectool getEmailConfig <configName>

**Example**

ectool getEmailConfig EmailConfig_test

**getEmailConfigs**

Retrieves all email configurations.
### Arguments

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

### Positional arguments

None

### Response

- **Returns**
  - 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:**

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

**Example**

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

### ectool

**syntax:**

```bash
ectool getEmailConfigs
```

**Example**

```bash
ectool getEmailConfigs
```

### modifyEmailConfig

Modifies an existing email configuration.

You must specify the `configName`.

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

**Positional arguments**

**configName**

**Response**

None or a status OK message.

**ec-perl**

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

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

tool

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

Example

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

Back to Top

API Commands - Email Notifier Management

- `createEmailNotifier` on page 255
- `createEventSubscription` on page 259
- `deleteEmailNotifier` on page 261
- `deleteEventSubscription` on page 264
- `getEmailNotifier` on page 266
- `getEmailNotifiers` on page 268
- `getEventSubscription` on page 270
- `getEventSubscriptions` on page 272
- `modifyEmailNotifier` on page 274
- `modifyEventSubscription` on page 278
- `sendEmail` on page 281

createEmailNotifier

Creates an email notifier attached to the specified object, such as a job, job step, project, procedure, application process or process step, component process or process step, or a workflow.

You must specify a `notifierName` and object locators for a job, job step, procedure, or procedure step.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(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</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>componentName</td>
<td>(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</td>
</tr>
<tr>
<td>condition</td>
<td>(Optional) Only send mail if the condition evaluates to &quot;true&quot;. The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is &quot;false&quot; or &quot;0&quot;. If no condition is specified, the notification is ALWAYS sent. Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) If specified, this argument must be the name of an emailConfig object. If it is not specified, the default is the name of the first emailConfig object defined for the ElectricFlow server (emailConfig objects are &quot;ordered&quot; ElectricFlow entities). Note: When using this argument, you must also include the formattingTemplate or the formattingTemplateFile argument. Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, `&lt;ul&gt; Argument type: String</td>
</tr>
<tr>
<td>destinations</td>
<td>(Optional) A mandatory argument for a create operation. A space-separated list of valid email addresses, email aliases, or ElectricFlow user names, or a string subject to property expansion that expands into such a list. Argument type: String</td>
</tr>
<tr>
<td>environmentNames</td>
<td>(Optional) Names of the environments. Argument type: Collection</td>
</tr>
<tr>
<td>eventType</td>
<td>(Optional) Use one of these values: `&lt;onStart</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>formattingTemplate</td>
<td>(Optional) A template for formatting email messages when an event [notification] is triggered by the emailNotifier. Make sure the content is formatted correctly, such as no illegal characters or spacing. Argument type: String</td>
</tr>
<tr>
<td>formattingTemplateFile</td>
<td>(Optional) This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the formatting template file is read and stored in the &quot;formatting template&quot; field. This is an alternative argument for --formattingTemplate and is useful if the &quot;formatting template&quot; field spans multiple lines.</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) Type of the gate. Argument type: GateType</td>
</tr>
<tr>
<td>groupNames</td>
<td>(Optional) A list of groups that receive the notification. Argument type: Collection</td>
</tr>
<tr>
<td>jobId</td>
<td>(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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>notificationType</td>
<td>(Optional) The notification type that is stored to the ec_notificationType property. Argument type: NotificationType</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline when a credential attached to a stage task.</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure. When using this argument, you must also enter the projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process that contains of the email notifier. Argument type: String</td>
</tr>
</tbody>
</table>
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step that contains of the email notifier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project. When using this argument, you must also enter the procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage when a credential is attached to a stage task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step. When using this argument, you must also enter projectName and procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>userNames</td>
<td>(Optional) The names of the users who receive the notification.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Collection</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

notifierName

### Response

None or status OK message.

### ec-perl

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

```perl
$cmdr->createEmailNotifier("testNotifier",
    {eventType => "onStart",
    condition => "$[/javascript if(myJobStep.outcome == 'warning') 'true'; else 'false';]",
    destinations => 'user1@abc.com user2@abc.com emailAlias1@abc.com',
    configName => "testConfiguration",
    projectName => "Project_test",
    procedureName => "Procedure_test",
```

ectool

**syntax:** ectool createEmailNotifier <notifierName> ...

Example

```bash
ectool createEmailNotifier testNotifier --condition "$[/javascript if(myJobStep.outcome == 'warning') 'true'; else 'false';]"
    --destinations "user1@abc.com user2@abc.com emailAlias1@abc.com"
    --projectName Project_test
    --procedureName Procedure_test
    --description "This is a test email notifier for Job completion"
```

createEventSubscription

Creates a list of event subscriptions.

You must specify a notifierName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td><strong>Descriptions</strong></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>componentName</td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) The type of the gate. Argument type: GateType</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>groupNames</td>
<td>A list of groups that receive the notification. Argument type: Collection</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>jobId</td>
<td>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</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process. Argument type: String</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step. Argument type: String</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project. When using this argument, you must also enter procedureName. Argument type: String</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition. Argument type: String</td>
</tr>
</tbody>
</table>
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 projectName and procedureName. Argument type: String

taskName | (Optional) The name of the task. 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**

None or status OK message.

**ec-perl**

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

*Example*

```
$cmdr->createEventSubscription ("testNotifier", {applicationName => "myApp"});
```

**ectool**

*Syntax:* `ectool createEventSubscription <notifierName> ...`

*Example*

```
ectool createEventSubscription testNotifier --applicationName myApp
```

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier that you want to delete. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>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</td>
</tr>
<tr>
<td>componentName</td>
<td>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</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) The type of the gate. Argument type: GateType</td>
</tr>
<tr>
<td>jobId</td>
<td>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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure. When using this argument, you must also enter <code>projectName</code>. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>Name of the process. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>Name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project. Argument type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
stageName  | (Optional) The name of the stage definition. Argument type: String
stateDefinitionName | Name of the state definition. Argument type: String
stateName | Name of the state. Argument type: String
stepName | Name of the step. When using this argument, you must also enter projectName and procedureName. Argument type: String
taskName | (Optional) The name of the task. Argument type: String
workflowDefinitionName | The name of the workflow definition. Argument type: String
workflowName | The name of the workflow. Argument type: String

Positional arguments
notifierName

Response
None or a status OK message.

ec-perl

**syntax:** $cmdr-&gt;deleteEmailNotifier(<notifierName>, [ . . ]); **Example**

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

ectool

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

**Example**

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(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</td>
</tr>
<tr>
<td>componentName</td>
<td>(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</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) The type of the gate. Argument type: GateType</td>
</tr>
<tr>
<td>groupNames</td>
<td>(Optional) A list of groups that receive the notification. Argument type: Collection</td>
</tr>
<tr>
<td>jobId</td>
<td>(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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) Name of the process.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) Name of the process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project. When using this argument, you</td>
</tr>
<tr>
<td></td>
<td>must also enter procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) Name of the step. When using this argument, you</td>
</tr>
<tr>
<td></td>
<td>must also enter projectName and procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>userNames</td>
<td>(Optional) A list of names of the users who receives the notification.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Collection</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**
- notifierName

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application that is related to the target email container. The email notifier is attached to a process or process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component that is related to the target email container. The email notifier is attached to a process or process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) Type of the gate.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step that is assigned automatically when the job step is created.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>Name of the process. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>Name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>Name of the step. When using this argument, you must also enter projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

notifierName
Response

Returns one emailNotifier element.

ect-perl

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

**Example**

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

ectool

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

**Example**

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

getEmailNotifiers

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

You must specify one or more object locators.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>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</td>
</tr>
<tr>
<td>componentName</td>
<td>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</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) Type of the gate. Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>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</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>Name of the process. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>Name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project. When using this argument, you must also enter procedureName. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>Name of the step. When using this argument, you must also enter projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow. Argument type: String</td>
</tr>
</tbody>
</table>
**Positional arguments**

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

**Response**

Returns one or more emailNotifier elements.

**ec/perl**

*Syntax:* $cmdr->getEmailNotifiers({<optionals>});

**Example**

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

**ectool**

*Syntax:* ectool getEmailNotifiers ...

**Example**

```ectool
ectool getEmailNotifiers --projectName Project_test
  --procedureName Procedure_test
```

**getEventSubscription**

Get an event subscription for the specified user or group.

You must specify a notifierName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application that is related to the target email container.</td>
</tr>
<tr>
<td></td>
<td>The email notifier is attached to a process or process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component that is related to the target email container.</td>
</tr>
<tr>
<td></td>
<td>The email notifier is attached to a process or process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) Type of the gate.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group that receives the notification. Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>Name of the process. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>Name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project. When using this argument, you must also enter procedureName. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>Name of the step. When using this argument, you must also enter projectName and procedureName. Argument type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
**taskName** | (Optional) The name of the task. Argument type: String
**userNames** | The names of the users who receives the notification. Argument type: String
**workflowDefinitionName** | The name of the workflow definition. Argument type: String
**workflowName** | The name of the workflow. Argument type: String

**Positional arguments**

- **notifierName**

**Response**

Returns an event subscription for a user or group.

**ec-perl**

*Syntax:* 
$cmdr->getEventSubscription (<notifierName>, {<optionals>});

*Example*

$cmdr->getEventSubscription("Error", {groupName => "Dev");

**ectool**

*Syntax:* 
ectool getEventSubscription <notifierName> ...

*Example*

ectool getEventSubscription Error --groupName "Dev"

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

**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> ...

*Example*

```
ectool getEventSubscriptions Error --applicationName "Pet Store"
```

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

You must specify a notifierName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(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.</td>
</tr>
<tr>
<td>componentName</td>
<td>(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.</td>
</tr>
<tr>
<td>condition</td>
<td>(Optional) Only send mail if the condition evaluates to &quot;true.&quot; The condition is a string subject to property expansion. Notification will not be sent if the expanded string is &quot;false&quot; or &quot;0&quot;. If no condition is specified, the notification is always sent.</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) If specified, this argument must be the name of an emailConfig object. If it is not specified, the default is the name of the first emailConfig object defined for the ElectricFlow server. (emailConfig objects are &quot;ordered&quot; ElectricFlow entities). Note: When using this argument, you must also include the formattingTemplate or the formattingTemplateFile argument.</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>destinations</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> This argument is mandatory for the &quot;create&quot; operation.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentNames</td>
<td>(Optional) The names of the environments.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Collection</td>
</tr>
<tr>
<td>eventType</td>
<td>(Optional) Use one of these values: `&lt;onStart</td>
</tr>
<tr>
<td></td>
<td>Argument type: EventType</td>
</tr>
<tr>
<td>formattingTemplate</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>formattingTemplateFile</td>
<td>(Optional) <strong>This option is supported only in Perl and ectool bindings - it is not part of the XML protocol.</strong> Contents of the <code>formatting template file</code> is read and stored in the &quot;formatting template&quot; field. This is an alternative argument for <code>formattingTemplate</code> and is useful if the &quot;formatting template&quot; field spans multiple lines.</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) Type of the gate.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupNames</td>
<td>(Optional) The list of the groups that receives the notification.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Collection</td>
</tr>
<tr>
<td>jobId</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String</td>
</tr>
<tr>
<td>newName</td>
<td>(Optional) New name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>notificationType</td>
<td>(Optional) The notification type that is stored to the ec_notificationType property. Argument type: NotificationType</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure. When using this argument, you must also enter the projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process that contains the email notifier. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step that contains the email notifier. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project. When using this argument, you must also enter the projectName. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) Name of the step. When using this argument, you must also enter projectName and procedureName. Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
<th>Argument type</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
<td>String</td>
</tr>
<tr>
<td>userNames</td>
<td>(Optional) The names of the users who receive the notification.</td>
<td>Collection</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
<td>String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
<td>String</td>
</tr>
</tbody>
</table>

#### Positional arguments

notifierName

#### Response

None or a status OK message.

#### ec-perl

**syntax:**

```perl
$cmdr->modifyEmailNotifier(<notifierName>, [<optionals>]);
```

**Example**

```perl
$cmdr->modifyEmailNotifier("testNotifier",
    {eventType => "onCompletion",
    projectName => "Project_test",
    procedureName => "Procedure_test",});
```

#### ectool

**syntax:**

```bash
ectool modifyEmailNotifier <notifierName> ...
```

**Example**

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

### modifyEventSubscription

**Modifies a list of event subscriptions.**

**You must specify a notifierName.**
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application that is related to the target email container. The email notifier is attached to a process or process step.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component that is related to the target email container. The email notifier is attached to a process or process step.</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) Type of the gate.</td>
</tr>
<tr>
<td>groupNames</td>
<td>The list of the groups that receives the notification.</td>
</tr>
<tr>
<td>jobId</td>
<td>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.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The unique identifier for a job step that is assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notificationType</td>
<td>The notification type that is stored to the ec_notificationType property.</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure. When using this argument, you must also enter the projectName.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process that contains of the email notifier.</td>
</tr>
</tbody>
</table>
## Arguments

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

### Positional arguments

- **notifierName**

### Response

Returns an event subscription for a user or group.

### ec-perl

**syntax:**

```perl
$cmdr->modifyEventSubscription (<notifierName>, [<optionals>]);
```

**Example**

```perl
$cmdr->modifyEventSubscription("Error", {componentName => "Config files"});
```
**ectool**

* syntax: ectool modifyEventSubscription <notifierName> ...

* 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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>attachment</td>
<td>One or more client-side files to send as attachments. The filename extension is examined to determine the content type. You can enter this argument more than once to specify multiple attachments.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Collection</td>
</tr>
<tr>
<td>bcc</td>
<td>A &quot;Bcc&quot; 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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Collection</td>
</tr>
<tr>
<td>cc</td>
<td>A &quot;Cc&quot; 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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Collection</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the email configuration. If no configuration is specified, the configuration named &quot;default&quot; is used.</td>
</tr>
</tbody>
</table>

*Note:* The user must have `execute` permission on the configuration.

Argument type: String
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>header</td>
<td>An RFC822 email header line (for example: &quot;reply-to: <a href="mailto:user@host.com">user@host.com</a>&quot;). This option can be specified multiple times. Argument type: Collection</td>
</tr>
<tr>
<td>html</td>
<td>The body of a simple HTML message. Argument type: String</td>
</tr>
<tr>
<td>htmlFile</td>
<td>Reads the specified client-side file and uses it as the body of a simple HTML message.</td>
</tr>
<tr>
<td>inline</td>
<td>Inline attachments in this format: <code>&lt;contentId&gt;=&lt;fileName&gt; [&lt;contentId&gt;=&lt;fileName&gt; ...]</code>. One or more inline attachments specified as a contentId and a client-side filename. The filename extension is examined to determine the content-type. The contentId can be referenced in an HTML body using the <code>cid:protocol</code>. For example: <code>&lt;img src=cid:myImage&quot; could reference &quot;--inline myImage=image.jpg</code> You can enter this argument more than once to add multiple attachments. Argument type: Collection</td>
</tr>
<tr>
<td>mailFrom</td>
<td>The email address used in the &quot;From&quot; header to use when sending ElectricFlow notification email. When the email configuration is specified, this value overrides the value in the configuration. Argument type: String</td>
</tr>
<tr>
<td>mailHost</td>
<td>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. Argument type: String</td>
</tr>
<tr>
<td>mailPort</td>
<td>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. Argument type: Integer</td>
</tr>
<tr>
<td>mailProtocol</td>
<td>The name of the mail protocol that must be SMTP or SMTPS. When the email configuration is specified, this value overrides the value in the configuration. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mailUser</td>
<td>The name of the email user for whom ElectricFlow sends email notifications. ElectricFlow also uses it when authenticating to the mail server. When the email configuration is specified, this value overrides the value in the configuration. Argument type: String</td>
</tr>
<tr>
<td>mailUserPassword</td>
<td>The password that ElectricFlow uses when authenticating to the mail server. When the email configuration is specified, this value overrides the value in the configuration. Argument type: String</td>
</tr>
</tbody>
</table>
| multipartMode      | The multipart message mode: <none|mixed|related|mixedRelated>  
|                    |   - none—Non-multipart message  
|                    |   - mixed—Single-root multipart element of type "mixed". Texts, inline elements, and attachments will be added to this root element.  
|                    |   - related—Multipart message with a single root multipart element of "related" type. Texts, inline elements, and attachments will be added to this root element. It works on most mail clients, except Lotus Notes.  
|                    |   - mixedRelated—Multipart "mixed" element with a nested multipart "related" element. Texts and inline elements will be added to the nested "related" element, while attachments will be added to the "mixed' root element. It works on most mail clients other than Mac Mail and some situations on Outlook. If you experience problems, use the "related" value.  
| raw                | A raw RFC 822 email message including headers to use as the basis for the email message. Additional options can be applied to this message. Argument type: String                                                 |
| rawFile            | Reads the specified client-side file and uses it as the entire mail message, including headers.                                                                                                         |
| subject            | The subject of the email message. Argument type: String                                                                                                                                           |
Arguments | Descriptions
--- | ---
text | The body of a simple text message. Argument type: String
textFile | Reads the specified client-side file and uses it as the body of a simple text message.
to | A "To" recipient for the email message. The recipient can be a user, group name or complete email address. You can enter this argument more than once to specify multiple recipients. Argument type: Collection

Positional arguments
None

Response
None or status OK message.

ec-perl

*Syntax:* $cmdr-&gt;sendEmail

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

**Example**

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

ectool

*Syntax:* ectool sendEmail

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

**Examples**

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

Back to Top

API Commands - Environment

createEnvironment on page 285
createEnvironmentInventoryItem on page 286
deleteEnvironment on page 288
deleteEnvironmentInventoryItem on page 289
getEnvironment on page 290
getEnvironmentApplications on page 290
getEnvironmentInventory on page 291
getEnvironmentInventoryItem on page 292
getEnvironmentInventoryItems on page 293
getEnvironments on page 293
modifyEnvironment on page 294
modifyEnvironmentInventoryItem on page 295

createEnvironment

Creates a new environment.

Required Arguments

projectName

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

  Argument Type: String

evironmentName

  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", "aEnv", {environmentEnabled => "true", description => "aDescription"});

ectool

Syntax:

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

Example:

ectool createEnvironment default newEnv --environmentEnabled true --description exampleText

createEnvironmentInventoryItem

Creates a new environment inventory item.

Required Arguments

projectName

Description: Name for the project 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

artifactVersion

Description: Artifact version 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

description

Description: Comment text describing this object; not interpreted by ElectricFlow.
Argument Type: String

status

Description: Status of the item.
Argument Type: String

Response

Returns an environment inventory item.

c-ec perl

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

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

ectool
Syntax:
ectool createEnvironmentInventoryItem <projectName> <environmentName> <applicationName> <componentName> <resourceName> [optionals...]

Example:
ectool createEnvironmentInventoryItem Default aEnv App1 ComponentA ResourceA -- artifactName Artifact1 --artifactVersion V3 --description aDescription

deleteEnvironment

Deletes an environment.

Required Arguments

projectName

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

environmentName

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

Optional Arguments

None

Response

None or a status OK message.

cp-perl
Syntax:
$<object>-deleteEnvironment(<projectName>, <environmentName>);

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

ectool
Syntax:
ectool deleteEnvironment <projectName> <environmentName>

Example:
ectool deleteEnvironment default envToDelete
deleteEnvironmentInventoryItem

Delete an inventory item.

Required Arguments

projectName

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

evironmentName

Description: Name of the environment.
Argument Type: String

applicationName

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

componentName

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

resourceName

Description: Name of the resource where the item is installed.
Argument Type: String

Optional Arguments

None

Response

None or a status OK message.

ec-perl

Syntax:

```
$startObject-&gt;deleteEnvironmentInventoryItem($projectName, $environmentName, $applicationName, $componentName, $resourceName);
```

Example:

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

ectool

Syntax:

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

Example:

```
ectool deleteEnvironmentInventoryItem "Default" "Env1A" "AppTest1" "Component1" "Server1"
```
getEnvironment

Retrieves an environment by name.

**Required Arguments**

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

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

**Optional Arguments**
None

**Response**
Retrieves an environment element.

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

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

getEnvironmentApplications

Retrieves a list of applications installed on the given environment.

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

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

**Optional Arguments**
None
Response
Retrieves a list of applications for the specified environment.

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

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

ectool
Syntax:
ectool getEnvironmentApplications <projectName> <environmentName>

Example:
ectool getEnvironmentApplications default newEnv

getEnvironmentInventory
Retrieves a per-component grouped list of inventory items.

Required Arguments

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

environmentName
Description: Name of the environment.
Argument Type: String

Optional Arguments
None

Response
Retrieves a per-component grouped list of inventory items.

ec-perl
Syntax:
$<object>--getEnvironmentInventory(<projectName>, <environmentName>, <applicationName>);

Example:
$ec->getEnvironmentInventory("Default", "aEnv", "App1");
ectool
Syntax:
    ectool getEnvironmentInventory <projectName> <environmentName> <applicationName>
Example:
    ectool getEnvironmentInventory default newEnv App1

getEnvironmentInventoryItem
Retrieves an inventory item.

Required Arguments

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

evironmentName
    Description: Name of the environment.
    Argument Type: String

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

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

resourceName
    Description: Name of the resource where the item is installed.
    Argument Type: String

Optional Arguments

None

Response
Retrieves an inventory item.

ec-perl
Syntax:
    $<object>-getEnvironmentInventoryItem(<projectName>,
    <environmentName>, <applicationName>, <componentName>,
    <resourceName>);
Example:
    Sec->getEnvironmentInventoryItem("Default", "aEnv", "App1", "Component1", "Server1");
ectool
Syntax:
    ectool getEnvironmentInventoryItem <projectName> <environmentName>
    <applicationName> <componentName> <resourceName>
Example:
    ectool getEnvironmentInventoryItem default newEnv Appl Component1 Server1

getEnvironmentInventoryItems
Retrieves all inventory items for a given environment.

Required Arguments

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

environmentName
    Description: Name of the environment.
    Argument Type: String

Optional Arguments

None

Response
    Retrieves all inventory items for the specified environment.

ec-perl
Syntax:
    $<object>-getEnvironmentInventoryItems(<projectName>,
    <environmentName>);
Example:
    Sec->getEnvironmentInventoryItems("Default", "aEnv");

ectool
Syntax:
    ectool getEnvironmentInventoryItems <projectName> <environmentName>
Example:
    ectool getEnvironmentInventoryItems default newEnv

getEnvironments
Retrieves all environments in a project.

Required Arguments

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

**Argument Type:** String

**Optional Arguments**

None

**Response**

Retrieves zero or more environment elements.

**ec-perl**

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

    Example:
    $ec->getEnvironments("Default");

**ectool**

    Syntax:
    ectool getEnvironments <projectName>

    Example:
    ectool getEnvironments default

**modifyEnvironment**

Modifies an environment.

**Required Arguments**

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

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

**Optional Arguments**

    description
        **Description:** Comment text describing this object; not interpreted at all by ElectricFlow.
        **Argument Type:** String

    environmentEnabled
        **Description:** True to enable the environment.
        **Argument Type:** Boolean

    newName
        **Description:** New name for an existing object that is being renamed.
Modify an existing environment inventory item.

**Required Arguments**

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

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

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

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

**resourceName**
- **Description:** Name of the resource where the item is installed.
- **Argument Type:** String

**Argument Type:** String

**Response**
- Retrieves an updated environment element.

**ec-perl**

**Syntax:**

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

**Example:**

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

**ectool**

**Syntax:**

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

**Example:**

```
ectool modifyEnvironment default testEnv --newName modEnv
--description exampleText
```
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

status
   Description: Inventory deployment status.
   Argument Type: JobOutcome

Response
   Retrieves an updated environment inventory item.

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

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

ectool
   Syntax:
   ectool modifyEnvironmentInventoryItem <projectName> <environmentName>
   <applicationName> <componentName> <resourceName> <artifactName>
Example:

ectool modifyEnvironmentInventoryItem default testEnv Appl Component1 Server1 Artifact1 V3

API Commands - Environment Tier

- addResourcesToEnvironmentTier on page 297
- createEnvironmentTier on page 298
- deleteEnvironmentTier on page 299
- getEnvironmentTier on page 300
- getEnvironmentTiers on page 300
- modifyEnvironmentTier on page 301

addResourcesToEnvironmentTier

Adds resources to the specified 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 for the environment tier; must be unique among all tiers for 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", "newEnv", "envTier2", {resourceNames => "local"});

tool Syntax:
tool addResourcesToEnvironmentTier <projectName> <environmentName> <environmentTierName> [optionals...]

Example:
tool addResourcesToEnvironmentTier default newEnv envTier2 --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

Response

Returns an environment tier element.

c-perl Syntax:

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

Example:
**createEnvironmentTier**

$ect->createEnvironmentTier("Default", "newEnv", "envTier2", {description => "Description"});

**ectool**

**Syntax:**

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

**Example:**

`ectool createEnvironmentTier default newEnv envTier1 --description exampleText`

---

**deleteEnvironmentTier**

Deletes an environment tier.

**Required Arguments**

**projectName**

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

**Argument Type:** String

**environmentName**

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

**Argument Type:** String

**environmentTierName**

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

**Argument Type:** String

**Optional Arguments**

None

**Response**

None or a status OK message.

---

**ec-perl**

**Syntax:**

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

**Example:**

`$ect->deleteEnvironmentTier("Default", "newEnv", "tierToDelete");`

**ectool**

**Syntax:**

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

**Example:**
getEnvironmentTier

Retrieves an environment tier by name.

Required Arguments

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

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

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

Optional Arguments

- None

Response

- Retrieves an environment tier element.

**ec-perl**

**Syntax:**

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

**Example:**

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

**ectool**

**Syntax:**

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

**Example:**

```
ectool getEnvironmentTier default newEnv envTier1
```

getEnvironmentTiers

Retrieves all environment tiers in an environment.

Required Arguments

- **projectName**
ElectricFlow Perl API Commands

modifyEnvironmentTier

Modifies an environment tier.

Required Arguments

projectName

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

Argument Type: String

environmentName

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

Argument Type: String

e-perl

Syntax:

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

Example:

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

ectool

Syntax:

ectool getEnvironmentTiers <projectName> <environmentName>

Example:

ectool getEnvironmentTiers default newEnv
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 object that is being renamed.
Argument Type: String

Response

Retrieves an updated environment tier element.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

ectool modifyEnvironmentTier default newEnv envTier1 --description new_exampleText ext --newName envTierA

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.

dc-perl

Syntax:

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

Example:

$ec->removeResourcesFromEnvironmentTier("Default", "Production", "Apache");

cctool

Syntax:

cctool removeResourcesFromEnvironmentTier <projectName> <environmentName> <environmentTierName> [optionals ...]

Example:

cctool removeResourcesFromEnvironmentTier "Default" "Production" "Apache"

API Commands - Gateway and Zone Management

createGateway on page 303

deleteGateway on page 305

getGateway on page 306

getGateways on page 307

modifyGateway on page 307

createZone on page 309

deleteZone on page 310

getZone on page 311

getZones on page 311

modifyZone on page 312

createGateway

Creates a new gateway.

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

You must specify **gatewayName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayName</td>
<td>The name of the gateway that must be unique among all gateway names.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: `&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tt&gt; &lt;th&gt; &lt;td&gt; &lt;tr&gt; &lt;ul&gt;.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>gatewayDisabled</td>
<td>(Optional) `&lt;Boolean flag&gt; 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>hostName1</td>
<td>(Optional) The domain name or IP of the agent where Resource1 resides. This host name is used by Resource2 to communicate with Resource1. Do not specify this option is you want to use the host name from the Resource1 definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>hostName2</td>
<td>(Optional) The domain name or IP of the agent where Resource2 resides. This host name is used by Resource1 to communicate with Resource2. Do not specify this option is you want to use the host name from the Resource2 definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>port1</td>
<td>(Optional) The port number used by Resource1. The default is the port number used by the resource.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>port2</td>
<td>(Optional) The port number used by Resource2. The default is the port number used by the resource.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>resourceName1</td>
<td>(Optional) Name of first resource in the gateway specification. Do not include spaces in the resource name. Other resources in this resource's zone forward messages through this resource to agents in the resourceName2 zone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
resourceName2 | (Optional) Name of second resource in the gateway specification. Do not include spaces in the resource name. Other resources in this resource's zone forward messages through this resource to agents in the resourceName1 zone.

**Positional arguments**
gatewayName

**Response**
Returns a gateway object.

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

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

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

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

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

**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-&gt;deleteGateway (<gatewayName>);

Example

$cmdr-&gt;deleteGateway ("AB_Gateway");

ectool

syntax: ectool deleteGateway <gatewayName>

Example

ectool deleteGateway "AB_Gateway"

getGateway

Finds a gateway by name.

You must specify a gatewayName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>gatewayName</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

Positional arguments

gatewayName

Response

Returns one gateway element.

ec-perl

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

Example

$cmdr-&gt;getGateway ("AB_Gateway");

ectool

syntax: ectool getGateway <gatewayName>
**getGateways**

Retrieves all gateways.

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

**Positional arguments**

None.

**Response**

Returns one or more gateway elements.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* `ectool getGateways`

*Example*

```bash
ectool getGateways
```

**modifyGateway**

Modifies an existing gateway.

You must specify a `gatewayName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| `gatewayName`| The name of the gateway.  
<p>|              | Argument type: String |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt;</code> <code>&lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code>. Argument type: String</td>
</tr>
<tr>
<td>gatewayDisabled</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>hostName1</td>
<td>The domain name or IP address of the agent where Resource1 resides. This host name is used by Resource2 to communicate with Resource1. Do not specify this option is you want to use the host name from the Resource1 definition. Argument type: String</td>
</tr>
<tr>
<td>hostName2</td>
<td>The domain name or IP address of the agent where Resource2 resides. This host name is used by Resource1 to communicate with Resource2. Do not specify this option is you want to use the host name from the Resource2 definition. Argument type: String</td>
</tr>
<tr>
<td>newName</td>
<td>New name of the gateway. Argument type: String</td>
</tr>
<tr>
<td>port1</td>
<td>The port number used by Resource1. The default is the port number used by the resource. Argument type: Integer</td>
</tr>
<tr>
<td>port2</td>
<td>The port number used by Resource2. The default is the port number used by the resource. Argument type: Integer</td>
</tr>
<tr>
<td>resourceName1</td>
<td>Name of first resource in the gateway specification. Do not include spaces in the resource name. Other resources in this resource’s zone forward messages through this resource to agents in the resourceName2 zone. Argument type: String</td>
</tr>
<tr>
<td>resourceName2</td>
<td>Name of second resource in the gateway specification. Do not include spaces in the resource name. Other resources in this resource’s zone forward messages through this resource to agents in the resourceName1 zone. Argument type: String</td>
</tr>
</tbody>
</table>
## Positional arguments

**gatewayName**

## Response

An updated gateway object.

### ec-perl

**syntax:**

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

**Example**

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

### ectool

**syntax:**

```
ectool modifyGateway <gatewayName> ...
```

**Example**

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

---

### createZone

Creates a new zone.

You must specify a `zoneName`.

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

Returns a zone object.

ec-perl

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

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

ectool

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

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

Back to Top

deleteZone

Deletes an existing zone.

You must specify a zoneName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoneName</td>
<td>The name of the zone to delete.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**
zoneName

Response

None

ec-perl

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

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

ectool

**syntax:**
```
ectool deleteZone <zoneName>
```
**Example**

ectool deleteZone DevZone

**getZone**

Finds a zone by name.

You must specify a zoneName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoneName</td>
<td>The name of the zone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

zoneName

**Response**

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

**ec-perl**

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

*Example*

$cmdr->getZone ("DevZone");

**ectool**

*syntax:* ectool getZone <zoneName>

*Example*

ectool getZone DevZone

**getZones**

Retrieves all zones.

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

**Response**
Returns a zone object.

**ec-perl**

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

*Example*

$cmdr->getZones();

**ectool**

*Syntax:* ectool getZones

*Example*

ectool getZones

---

**modifyZone**

Modifies an existing zone.

You must specify a `zoneName`.

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

**Positional arguments**

`zoneName`

**Response**

Returns an updated zone element.
**ec-perl**

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

*Example*

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

**ectool**

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

*Example*

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

---

**API Commands - Job Management**

- [abortAllJobs](#) on page 314
- [abortJob](#) on page 315
- [abortJobStep](#) on page 316
- [deleteJob](#) on page 317
- [findJobSteps](#) on page 317
- [getJobDetails](#) on page 319
- [getJobInfo](#) on page 320
- [getJobNotes](#) on page 320
- [getJobs](#) on page 321
- [getJobsForSchedule](#) on page 323
- [getJobStatus](#) on page 323
- [getJobStepDetails](#) on page 324
- [getJobStepStatus](#) on page 325
- [getJobSummaries](#) on page 326
- [getJobSummary](#) on page 328
- [moveJobs](#) on page 329
- [runProcedure](#) on page 330
- [setJobName](#) on page 333

**External Job APIs**

- [completeJob](#) on page 335
- [completeJobStep](#) on page 336
- [createJob](#) on page 338
**abortAllJobs**

Aborts all running jobs.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>reason</td>
<td>A string added to the aborted job/jobstep that describes or records the reason for the abort. The server records this value, but places no meaning on the string, which is similar to a text description. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

None or status OK message.

**ec-perl**

*Syntax:* $cmdr->abortAllJobs({...});

*Example*

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

**ectool**

*Syntax:* ectool abortAllJobs ...

*Example*

ectool abortAllJobs --force 1
**abortJob**

Aborts a running job.

You must enter a `jobId`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job 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</td>
</tr>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>reason</td>
<td>A string added to the aborted job/jobstep that describes or records the reason for the abort. The server records this value, but places no meaning on the string, which is similar to a text description. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

jobId

**Response**

None or status OK message.

**ec-perl**

* syntax:*

```perl
$cmdr->abortJob(<jobId>, {...});
```

* Example*

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

**ectool**

* syntax:*

```bash
ectool abortJob <jobId> ...
```

* Example*

```bash
ectool abortJob 4fa765dd-73f1-11e3-b67e-b0a420524153 --force 1
```
# abortJobStep

Aborts any type of 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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobStepId</td>
<td>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</td>
</tr>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>reason</td>
<td>A string added to the aborted job/jobstep that describes or records the reason for the abort. The server records this value, but places no meaning on the string, which is similar to a text description. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- jobStepId

**Response**

None or status OK message.

**ec-perl**

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

*Example*

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

**ectool**

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

*Example*

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

[Back to Top]
deleteJob

Deletes a job from the ElectricFlow database.

You must specify a jobId.

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

**Positional arguments**

jobId

**Response**

None or a status OK message.

**ec-perl**

*syntax:* $cmdr->deleteJob(<jobId>);

*Example*

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

**ectool**

*syntax:* ectool deleteJob <jobId>

*Example*

ectool deleteJob 4fa765dd-73f1-11e3-b67e-b0a420524153

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.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job 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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>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</td>
</tr>
<tr>
<td>filter</td>
<td>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</td>
</tr>
<tr>
<td>numObjects</td>
<td>&lt;full object count&gt; This specifies the number of full job steps (not just the IDs) returned in the response. Returned job steps will be from the beginning of the list. If numObjects is not specified, all job steps in the list of object IDs are returned. Any and all job steps can be retrieved using the getObjects command. Argument type: Integer</td>
</tr>
<tr>
<td>selects</td>
<td>This is an unordered list of property names that specify additional top-level properties to return for each object. See the code example for findObjects for instructions on forming the list and passing it to the ElectricFlow Perl API. Argument type: Collection</td>
</tr>
</tbody>
</table>

**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.

**getJobDetails**

Retrieves complete information about a job, including details from each job step.

You must specify a **jobId**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>jobId</strong></td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job 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</td>
</tr>
<tr>
<td><strong>structureOnly</strong></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**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> ...
getJobInfo

Retrieves all information about a job, except for job step information. You must specify a jobId.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job 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</td>
</tr>
</tbody>
</table>

**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
```

getJobNotes

Retrieves the notes property sheet from a job. You must specify a jobId.
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job 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</td>
</tr>
</tbody>
</table>

#### Positional arguments

**jobId**

#### Response

A [propertySheet](#) element that contains the job.

**ec-perl**

*Syntax:* $cmdr->getJobNotes(<jobId>);

*Example*

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

**ectool**

*Syntax:* ectool getJobNotes <jobId>

*Example*

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

---

### getJobs

Retrieves summary information for a list of jobs. By default, all jobs are returned.

**Notes:**

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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstResult</td>
<td>The first row to return in the results. This is the <code>&lt;index number&gt;</code> 0-based index that identifies the first element returned from filtered, sorted result set. Argument type: Integer</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>maxResults</td>
<td>This number, <code>&lt;result count&gt;</code>, sets the maximum number of returned jobs.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>sortKey</td>
<td>How to sort the results: `&lt;jobId</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>sortOrder</td>
<td>The order in which to sort the results: `&lt;ascending</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>status</td>
<td>Filter jobs by this status: `&lt;running</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

None

### Response

One or more `job` elements. A `job` element contains summary information only.

### ec-perl

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

**Examples**

How do I get the first 10 jobs (index 0-9)?

```perl
$cmdr->getJobs({maxResults=>10});
```

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

```perl
$cmdr->getJobs({firstResult=>10, maxResults=>10});
```

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

```perl
$cmdr->getJobs({sortKey=>'start', sortOrder=>'descending', maxResults=>1});
```

### ectool

**syntax:** `ectool getJobs ...`

**Examples**

How do I get the first 10 jobs (index 0-9)?

`ectool getJobs --maxResults 10`

How do I get the next 10 jobs (index 10-19)?
ectool getJobs --firstResult 10 --maxResults 10

How do I get the most recent job by start time?
ectool getJobs --sortKey start --sortOrder descending --maxResults 1

getJobsForSchedule
Retrieves jobs started by a specific schedule.
You must specify a projectName and scheduleName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name for the schedule that must be unique among all schedules for the project.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, scheduleName

**Response**

Returns an XML stream containing any number of job elements. The job elements contain summary information only.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* ectool getJobsForSchedule <projectName> <scheduleName>

*Example*

ectool getJobsForSchedule Test ea1

getJobStatus
Retrieves the status of a job.
electric flow

You must specify the jobId.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job 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</td>
</tr>
</tbody>
</table>

**Positional arguments**

jobId

**Response**

Values for status and outcome as follows:

Possible values for status:
- pending - The job is not yet runnable—it is waiting for other steps to complete first. A job should not stay in this state for longer than a few seconds.
- runnable - The job is ready to run, but it is waiting for a resource to become available.
- running - The job is assigned to a resource and is executing the step command.
- completed - The job finished executing.

Possible values for outcome: The outcome is accurate only if the job status is "completed."
- success - The job finished successfully.
- warning - The job completed with no errors, but encountered some suspicious conditions.
- error - The job has finished execution with errors.

**ec-perl**

*Syntax:* $cmdr->getJobStatus(<jobId>);

*Example*

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

**ectool**

*Syntax:* ectool getJobStatus <jobId>

*Example*

ectool getJobStatus 4fa765dd-73f1-11e3-b67e-b0a420524153

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobStepId</td>
<td>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</td>
</tr>
<tr>
<td>structureOnly</td>
<td>&lt;Boolean flag - &lt;0</td>
</tr>
</tbody>
</table>

**Positional arguments**

jobStepId

**Response**

A jobStep element.

**ec-perl**

```perl
$scmdr->getJobStepDetails(<jobStepId>, {...});
```

**Example**

```perl
$scmdr->getJobStepDetails(5da765dd-73f1-11e3-b67e-b0a420524153);
```

**ectool**

```bash
ectool getJobStepDetails <jobStepId> ...
```

**Example**

```bash
ectool getJobStepDetails 5da765dd-73f1-11e3-b67e-b0a420524153
```

**getJobStepStatus**

Retrieves the status of a job step.

You must specify the jobStepId.
Arguments | Descriptions
--- | ---
jobStepId | The unique ElectricFlow-generated identifier (a UUID) for a job step that is assigned automatically when the job step is created. The system also accepts a job step name assigned to the job step by its name template. Argument type: UUID

**Positional arguments**

jobStepId

**Response**

A status tag - for example: `<status>completed</status>`

Possible values for status:

- pending - The job step is not yet runnable— it is waiting for other steps to complete first. A job should not stay in this state for longer than a few seconds.
- runnable - The job step is ready to run, but it is waiting for a resource to become available.
- running - The job step is assigned to a resource and is executing the step command.
- completed - The job step finished executing.

**ec-perl**

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

*Example*

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

**ectool**

*syntax:* `ectool getJobStepStatus <jobStepId>`

*Example*

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

**getJobSummaries**

Returns summary information about jobs.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>filters</td>
<td>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. Each element of the filter list is a hash reference containing one filter criterion. You may specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricFlow Perl API. Two types of filters: &quot;property filters&quot; - used to select objects based on the value of the object's intrinsic or custom property &quot;boolean filters&quot; (&quot;and&quot;, &quot;or&quot;, &quot;not&quot;) - used to combine one or more filters using boolean logic. Each &quot;property filter&quot; consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by ElectricFlow or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property. Property filter operators are: <code>between</code> (2 operands) <code>contains</code> (1) <code>equals</code> (1) <code>greaterOrEqual</code> (1) <code>greaterThan</code> (1) <code>in</code> (1) <code>lessOrEqual</code> (1) <code>lessThan</code> (1) <code>like</code> (1) <code>notEqual</code> (1) <code>notLike</code> (1) <code>isNotNull</code> (0) <code>isNull</code> (0) A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter. Boolean operators are: <code>not</code> (1 operand) <code>and</code> (2 or more operands) <code>or</code> (2 or more operands) Argument type: Collection</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
includeLastSuccess | `<Boolean flag - 0|1|true|false>`
When this argument is set to 1, the step will include the last successful job if it was not already included.
Argument type: Boolean
maxFailedSteps | Maximum number of failed steps to return.
Argument type: Integer
maxJobs | Maximum number of jobs to return.
Argument type: Integer
maxRunningSteps | 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 ( {...});`

*Example*

```perl
$cmdr->getJobSummaries ( {maxFailedSteps => 6} );
```

**ectool**

*Syntax*: `ectool getJobSummaries ...

*Example*

```bash
ectool getJobSummaries --maxFailedSteps 6
```

Back to Top

**getJobSummary**

Returns a job and its job steps with only the specified job and job step properties.
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
jobProperties | A comma-separated list of intrinsic job properties to include in the output. Argument type: String
jobStepProperties | 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
```

**moveJobs**

Moves jobs from one project to another project.

You must specify `sourceProject` and `destinationProject`.

Arguments | Descriptions
--- | ---
sourceProject | Name of the project that contains the jobs you want to move. Argument type: String
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationProject</td>
<td>New project that will contain the jobs.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

- sourceProject, destinationProject

### Response

- None or a status OK message.

### ec-perl

**syntax:**

```perl
$cmdr->moveJobs(<sourceProject>, <destinationProject>);
```

**Example**

```perl
$cmdr->moveJobs("ProjectA", "ProjectB");
```

### ectool

**syntax:**

```bash
ectool moveJobs <sourceProject> <destinationProject> ...
```

**Example**

```bash
ectool moveJobs "ProjectA" "ProjectB"
```

### runProcedure

Creates and starts a new job using a procedure directly or a procedure specified indirectly through a schedule.

Returns a new job ID. 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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>actualParameter</td>
<td>(Optional) Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure. Used in conjunction with procedureName to set the value of the actual parameters. Do not use this argument with scheduleName. Argument type: Map</td>
</tr>
<tr>
<td>credentials</td>
<td>(Optional) Use the following syntax to specify a credential: &lt;credName&gt;=&lt;userName&gt; [&lt;credName&gt;=&lt;userName&gt; ...]. Argument type: Collection</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td><em>(Optional)</em> credentialName can be one of two forms: <strong>relative</strong> <em>(for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object.</em> <strong>absolute</strong> <em>(for example, &quot;projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object’s project.</em></td>
</tr>
<tr>
<td>destinationProject</td>
<td><em>(Optional)</em> This argument is used to specify the name of the destination project.</td>
</tr>
<tr>
<td>password</td>
<td><em>(Optional)</em> The password for the user running the procedure.</td>
</tr>
<tr>
<td>priority</td>
<td><em>(Optional)</em> Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number. **Priority values are: low</td>
</tr>
<tr>
<td>procedureName</td>
<td><em>(Optional)</em> The name for the procedure; must be unique within the project.</td>
</tr>
<tr>
<td>scheduleName</td>
<td><em>(Optional)</em> 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.</td>
</tr>
<tr>
<td>userName</td>
<td><em>(Optional)</em> The name of the user who is running the procedure.</td>
</tr>
<tr>
<td>pollInterval</td>
<td><em>(Optional)</em> This argument requires setting a value in <em>seconds</em> to determine how often ectl queries the ElectricFlow server for job status, but this is not an indefinite activity—set the <em>timeout value</em> 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.</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
**timeout** | *(Optional)* This argument requires a value set in *seconds*. If `pollInterval` is specified, this `timeout` causes `runProcedure` to stop waiting for the job to complete. It does not stop the job itself. If `pollInterval` is used and `timeout` is not used, `pollInterval` will timeout in 60 seconds.

**Positional arguments**
- `projectName`

**Response**
The new job ID number.

**ec-perl**

*Syntax:* $cmdr-&gt;runProcedure(<project name>, {<optionals>});

*Example*

```
$cmdr-&gt;runProcedure("Sample Project", {procedureName =&gt; "Delay",
    actualParameter =&gt; {actualParameterName =&gt; "Delay Time", value =&gt; 10}});
$xpath = $ec-&gt;runProcedure("BSHTest",
    {procedureName =&gt; "FakeMotoBuild",
        actualParameter =&gt; [
            {actualParameterName =&gt; "builddir", value =&gt; $cwd},
            {actualParameterName =&gt; "board", value =&gt; $board},
            {actualParameterName =&gt; "myview", value =&gt; $cwv},
            {actualParameterName =&gt; "resourcetouse",
                value =&gt; $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"
```

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job 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</td>
</tr>
<tr>
<td>newName</td>
<td>New name of the job. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

*jobId, newName*

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* $cmdr->setJobName(<jobId>, <newName>);

*Examples*

$cmdr->setJobName(4fa765dd-73f1-11e3-b67e-b0a420524153, "Delay Test_541"); (from the command line)

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

**ectool**

*Syntax:* ectool setJobName <jobId> <newName> ...

*Examples*

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

ectool setJobName "TestJob"_252

**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 run time for the root step.

You must specify a jobId.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>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</td>
</tr>
</tbody>
</table>
| force     | <Boolean flag - 0|1|true|false> If true, all external steps belonging to the job will be marked "complete". This arguments determines whether all external steps under the job should be recursively marked "complete".  

**Note:** If this API is called on a job launched with runProcedure, there is no effect unless force is enabled, in which case only external steps are affected.  
Argument type: Boolean |
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| outcome   | Overall outcome for a job or step:  
|           |  
|           |  
|           | success - The job finished successfully.  
|           | warning - The job completed with no errors, but encountered some suspicious conditions.  
|           | error - The job has finished execution with errors.  
|           | If specified, the outcome overrides any previously propagated outcome value.  
|           | Argument type: JobOutcome |

### Positional arguments

**jobId**

### Response

None or status OK message.

### ec-perl

**syntax:** $cmdr->completeJob(<jobId>);

**Example**

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

### ectool

**syntax:** ectool completeJob <jobId>

**Example**

```
ectool completeJob 1234
```

### completeJobStep

Completes an externally managed job step. Marks the job step "completed" when all child steps are completed and updates the step run time.

You must specify a jobStepId.
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobStepId</td>
<td>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</td>
</tr>
<tr>
<td>exitCode</td>
<td>The exit code of a job step. Argument type: Integer</td>
</tr>
<tr>
<td>force</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>outcome</td>
<td>Overall outcome for a job or step:</td>
</tr>
<tr>
<td></td>
<td>- <code>success</code> - The job step finished successfully.</td>
</tr>
<tr>
<td></td>
<td>- <code>warning</code> - The job step completed with no errors, but encountered some suspicious conditions.</td>
</tr>
<tr>
<td></td>
<td>- <code>error</code> - The job step has finished execution with errors.</td>
</tr>
<tr>
<td></td>
<td>- <code>skipped</code> - The job step was skipped.</td>
</tr>
<tr>
<td></td>
<td>Argument type: JobOutcome</td>
</tr>
</tbody>
</table>

**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
```
**createJob**

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

You must specify `projectName` or `destinationProject`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationProject</td>
<td>If specified, determines the project where the job will reside. You must have <em>modify</em> 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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobNameTemplate</td>
<td>If specified, the job name will be generated by expanding this argument value. <code>Note</code>: The job name is generated by expanding the <code>jobNameTemplate</code> argument or the <code>jobNameTemplate</code> from the procedure or the system default.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>If specified, <code>projectName</code> and <code>procedureName</code> are used as a template for the job. You must have <em>execute</em> permission on the procedure. <code>Note</code>: 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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project where this job will reside. You must have <em>modify</em> 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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| status    | The starting status for the job: pending|runnable|scheduled|running>. The status argument determines the "starting" job status. This is useful if you want to immediately go into a specific status without having to use modifyJob to update the status. Defaults to pending. Possible values for status:  
- `pending`—The job is not yet runnable.  
- `runnable`—The job is ready to run.  
- `scheduled`—The job is scheduled to run.  
- `running`—The job is executing.  
Argument type: JobStatus |

### Positional arguments

None

### Response

The new job ID number.

### ec-perl

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

**Example**

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

### ectool

**syntax:** ectool createJob ...

**Example**

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

### 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.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the <em>called</em> procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure. For more information about parameters, click <a href="#">here</a>. Argument type: Map</td>
</tr>
<tr>
<td>alwaysRun</td>
<td>If set to 1, indicates this job step will run even if the job is aborted before the job step completes. A useful argument for running a &quot;cleanup&quot; step that should run whether the job step is successful or not. The value for <code>alwaysRun</code> is a <code>&lt;Boolean</code> flag - 0</td>
</tr>
<tr>
<td>broadcast</td>
<td>Use this flag to run the same job step on several resources at the same time. The job step is &quot;broadcast&quot; to all resources listed in the resourceName argument. The broadcast value = <code>&lt;Boolean</code> flag - 0</td>
</tr>
<tr>
<td>command</td>
<td>The command to run. This argument is applicable to command job steps only. Argument type: String</td>
</tr>
<tr>
<td>condition</td>
<td>If empty or non-zero, the job step will run. If set to &quot;0&quot;, the job step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the job steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>The credential to use for impersonation on the agent. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object’s project. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>credentials</td>
<td>Refers to one or more credentials to attach to this job step. These are &quot;dynamic&quot; credentials, captured when a job is created. Dynamic credentials are stored on the server temporarily until the job completes and then discarded. For more information about credentials, see the &quot;Credentials and User Impersonation&quot; topic in the ElectricFlow User Guide. Argument type: Collection</td>
</tr>
<tr>
<td>errorHandling</td>
<td>Determines what happens to the procedure if the step fails:</td>
</tr>
<tr>
<td></td>
<td>- failProcedure - The current procedure continues, but the overall status is error (default).</td>
</tr>
<tr>
<td></td>
<td>- abortProcedure - Aborts the current procedure, but allows already-running steps in the current procedure to complete.</td>
</tr>
<tr>
<td></td>
<td>- abortProcedureNow - Aborts the current procedure and terminates running steps in the current procedure.</td>
</tr>
<tr>
<td></td>
<td>- abortJob - Aborts the entire job, terminates running steps, but allows alwaysRun steps to run.</td>
</tr>
<tr>
<td></td>
<td>- abortJobNow - Aborts the entire job and terminates all running steps, including alwaysRun steps.</td>
</tr>
<tr>
<td></td>
<td>- ignore - Continues as if the step succeeded.</td>
</tr>
<tr>
<td>exclusive</td>
<td>If set to 1, indicates this job step should acquire and retain this resource exclusively. The value for exclusive is a &lt;Boolean flag -0</td>
</tr>
<tr>
<td>exclusiveMode</td>
<td>Use one of the following options:</td>
</tr>
<tr>
<td></td>
<td>- None - the &quot;default&quot;, which does not retain a resource.</td>
</tr>
<tr>
<td></td>
<td>- Job - keeps the resource for the duration of the job step. No other job can use this resource, regardless of its step limit, until this job completes or &quot;Release Exclusive&quot; is used in a job step. Future steps for this job will use this resource in preference to other resources—if this resource meets the needs of the job steps and its step limit is not exceeded.</td>
</tr>
<tr>
<td></td>
<td>- Step - keeps the resource for the duration of the job step.</td>
</tr>
<tr>
<td></td>
<td>- Call - keeps the resource for the duration of the procedure that called this job step, which is equivalent to 'job' for top level steps.</td>
</tr>
<tr>
<td></td>
<td>Argument type: ExclusiveMode</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>external</strong></td>
<td>If set, indicates this job step is an external step. ElectricFlow will not schedule or run agent commands for external steps, but instead, represents a step managed outside of ElectricFlow. The typical usage is with an external Job (see createJob). The status of an external job step is set using modifyJobStep, and it can be completed using completeJobStep. The value for external is a `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><strong>jobStepId</strong></td>
<td>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. If both jobStepId and parentPath are specified, parentPath is preferred. Argument type: UUID</td>
</tr>
<tr>
<td><strong>jobStepName</strong></td>
<td>The name of the job step. Argument type: String</td>
</tr>
<tr>
<td><strong>logFileName</strong></td>
<td>A custom log file name produced by running the job step. By default, ElectricFlow assigns a unique name for this file. Argument type: String</td>
</tr>
<tr>
<td><strong>parallel</strong></td>
<td>If set, indicates this job step should run at the same time as adjacent job steps marked to run as parallel also. The value for parallel is a `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><strong>parentPath</strong></td>
<td>The path of the parent job step. If both jobStepId and parentPath are specified, parentPath is preferred. Argument type: String</td>
</tr>
<tr>
<td><strong>postProcessor</strong></td>
<td>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 &quot;postp&quot; for this purpose. The value for postProcessor is a command string for invoking a post-processor program in the platform shell for the resource (cmd for Windows, sh for UNIX). Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| precondition  | The **precondition** property (if it exists) is copied to the job step when the step is created. When the job step is eligible to transition from pending to runnable, the precondition is evaluated. If the precondition result is empty, \texttt{false}, or \texttt{"0"}, the step remains in the pending state. Any other value allows the step to proceed to the runnable state. **Note:** A precondition property allows steps to be created with "pause", which then pauses the procedure. In a paused state, all currently running steps continue, but no additional steps will start. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a **precondition** is evaluated. A **precondition** is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a \texttt{"0"} or \texttt{"false"} is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE. **Argument type:** String  

| procedureName | The name of the procedure that will contain this job step. **Argument type:** String |

#### Precondition example:

Assume we defined these 4 steps:

1. Build object files and executables
2. Build installer
3. Run unit tests
4. Install bits on test system

Step 1 is an ordinary serial step.  
Steps 2 and 3 can run in parallel because they depend only on step 1's completion. 
Step 4 depends on step 2, but not step 3.

You can achieve optimal step execution order with preconditions:

- Make steps 2-4 run in parallel.
- Step 2 needs a job property set at the end of its step to indicate step 2 is completing  
  \texttt{(/myJob/buildInstallerCompleted=1)}.
- Set a precondition in step 4:  
  \$[/myJob/buildInstallerCompleted]
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that contains the procedure where you are adding a new job step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>releaseExclusive</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>&quot;exclusively&quot;. <strong>Note</strong>: Setting this flag to &quot;true&quot; is the same as setting releaseMode to release.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>releaseMode</td>
<td>Use one of the following options:</td>
</tr>
<tr>
<td></td>
<td>• none - the &quot;default&quot; - no action if the resource was not previously marked as &quot;retain.&quot;</td>
</tr>
<tr>
<td></td>
<td>• release - releases the resource at the end of this job step.</td>
</tr>
<tr>
<td></td>
<td>If the resource for the job step was previously acquired with &quot;Retain exclusive&quot; (either by this job</td>
</tr>
<tr>
<td></td>
<td>step or some preceding job step), the resource exclusivity is canceled at the end of this job step.</td>
</tr>
<tr>
<td></td>
<td>The resource is released in the normal way so it may be acquired by other jobs.</td>
</tr>
<tr>
<td></td>
<td>• releaseToJob - allows a job step to promote a &quot;step exclusive&quot; resource to a Job exclusive resource.</td>
</tr>
<tr>
<td></td>
<td>Argument type: ReleaseMode</td>
</tr>
<tr>
<td>resourceName</td>
<td>Name for the resource; must be unique among all resources.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>shell</td>
<td>Where <em>shell</em> is the name of a program used to execute commands contained in the &quot;command&quot; field. The</td>
</tr>
<tr>
<td></td>
<td>name of a temporary file containing commands will be appended to the end of this invocation line.</td>
</tr>
<tr>
<td></td>
<td>Normally, this file is a command shell, but it can be any other command line program. The default is</td>
</tr>
<tr>
<td></td>
<td>to use the standard shell for the platform it runs on (cmd for Windows, sh for UNIX). Applicable to</td>
</tr>
<tr>
<td></td>
<td>command steps only.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| status          | <pending|runnable|scheduled|running>  
> The `status` argument determines the "starting" job status. This is useful if you want to immediately go into a specific status without having to use `modifyJobStep` to update the status. Defaults to pending.  
> Possible values for `status`:  
> - pending—The job step is not yet runnable.  
> - runnable—The job step is ready to run.  
> - scheduled—The job step is scheduled to run.  
> - running—The job step is executing.  
> Argument type: JobStatus |
<p>| stepName        | The name of the new job step you are creating. You can use any name of your choice.                                                        |
|                 | Argument type: String                                                                                                                      |
| subprocedure    | The name of the nested procedure to call when this job step runs. If a subprocedure is specified, do not include the command or commandFile options. |
|                 | Argument type: String                                                                                                                      |
| subproject      | If a subprocedure argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.   |
|                 | Argument type: String                                                                                                                      |
| timeLimit       | The maximum length of time the job step is allowed to run. After the time specified, the job step will be aborted. The time limit is specified in units that can be hours, minutes, or seconds. |
|                 | Argument type: String                                                                                                                      |
| timeLimitUnits  | Specify <code>hours|minutes|seconds</code> for time limit units.                                                                                                                |
|                 | Argument type: TimeLimitUnits                                                                                                               |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>workingDirectory</td>
<td>The ElectricFlow agent sets this directory as the &quot;current working directory,&quot; 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.</td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If running a job step on a proxy resource, this directory must exist on the proxy target.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace where this job step log files will be stored.</td>
</tr>
</tbody>
</table>

**Positional arguments**

jobStepId or parentPath

**Response**

Returns a jobStep object.

**ec-perl**

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

**Examples**

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

$cmdr->createJobStep ({jobStepId => "5da765dd-73f1-11e3-b67e-b0a420524153", external => "1"});

# Create a job step that calls a subprocedure and passes a parameter credential
# 'coolProcedure' is a procedure within the Default project with one parameter
# credential named 'sshCredentialParameter'.
$cmdr->createJobStep(
    {
        projectName => 'Default',
        subprocedure => 'coolProcedure',
        actualParameter => [
            {
                actualParameterName => 'sshCredentialParameter',
                value => 'sshCredentialParameter'
            }
        ]
    },
)
credential => [ 
{
    credentialName => 'sshCredentialParameter',
    userName => 'sshUser',
    password => 'super_secure_sshPassword'
}
],
]);

# Create two parallel job steps and send them to the 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'
                },
                {
                    actualParameterName => 'input',
                    value => 'helloWorld'
                }
            ],
    ],
    $batch->createJobStep(
        {
            parallel => '1',
            projectName => 'Default',
            subprocedure => 'coolProcedure',
            actualParameter => [
                {
                    actualParameterName => 'input',
                    value => 'helloWorld'
                },
                {
                    actualParameterName => 'input',
                    value => 'helloWorld'
                }
            ],
    )};
actualParameter => [
  {
    actualParameterName => 'input',
    value => 'helloWorld'
  },
],
);

# Send off the requests
$batch->submit();

**ectool**

*syntax:* ectool createJobStep ...

**Examples**

ectool createJobStep --parentPath /jobs/123 --external 1

ectool createJobStep --jobStepId 5da765dd-73f1-11e3-b67e-b0a420524153 --external 1

ectool createJobStep --parallel 1 --projectName Default --subprocedure coolProcedure --actualParameter input=helloWorld

**modifyJob**

Modifies the status of an externally managed job.

You must specify a **jobId**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>jobId</strong></td>
<td>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</td>
</tr>
</tbody>
</table>
### Status

**Arguments**

<table>
<thead>
<tr>
<th>Status</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pending</td>
<td>The job is not yet runnable.</td>
</tr>
<tr>
<td>runnable</td>
<td>The job is ready to run.</td>
</tr>
<tr>
<td>scheduled</td>
<td>The job is scheduled to run.</td>
</tr>
<tr>
<td>running</td>
<td>The job is executing.</td>
</tr>
</tbody>
</table>

This argument determines the current status of the job, and also sets related timing values.

**Possible values for status:**

- pending – The job is not yet runnable.
- runnable – The job is ready to run.
- scheduled – The job is scheduled to run.
- running – The job is executing.

**Argument type:** JobStatus

### Positional arguments

- **jobId**

### Response

The jobId element.

#### ec-perl

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

**Example**

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

#### ectool

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

**Example**

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

### modifyJobStep

Modifies the status of an externally managed job step.

You must specify a jobStepId.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobStepId</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>Argument type:</strong> UUID</td>
</tr>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure.</td>
</tr>
<tr>
<td></td>
<td>For more information about parameters, click <a href="#">here</a>.</td>
</tr>
<tr>
<td></td>
<td><strong>Argument type:</strong> Map</td>
</tr>
<tr>
<td>alwaysRun</td>
<td>If set to 1, indicates this job step will run even if the job is aborted before the job step completes. A useful argument for running a &quot;cleanup&quot; step that should run whether the job step is successful or not. The value for alwaysRun is a `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td><strong>Argument type:</strong> Boolean</td>
</tr>
<tr>
<td>broadcast</td>
<td>Use this flag to run the same job step on several resources at the same time. The job step is &quot;broadcast&quot; to all resources listed in the resourceName argument. The broadcast value = `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td><strong>Argument type:</strong> Boolean</td>
</tr>
<tr>
<td>command</td>
<td>The command to run. This argument is applicable to command job steps only.</td>
</tr>
<tr>
<td></td>
<td><strong>Argument type:</strong> String</td>
</tr>
<tr>
<td>condition</td>
<td>If empty or non-zero, the job step will run. If set to &quot;0&quot;, the job step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the job steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
</tbody>
</table>
| errorHandling | Determines what happens to the procedure if the step fails:  
  - failProcedure - The current procedure continues, but the overall status is error (default).  
  - abortProcedure - Aborts the current procedure, but allows already-running steps in the current procedure to complete.  
  - abortProcedureNow - Aborts the current procedure and terminates running steps in the current procedure.  
  - abortJob - Aborts the entire job, terminates running steps, but allows alwaysRun steps to run.  
  - abortJobNow - Aborts the entire job and terminates all running steps, including alwaysRun steps.  
  - ignore - Continues as if the step succeeded.  
  
  Argument type: ErrorHandling |
| exclusive | If set to 1, indicates this job step should acquire and retain this resource exclusively. The value for exclusive is a <Boolean flag -0|1|true|false>. Defaults to "false".  
  
  Note: Setting exclusive, sets exclusiveMode to "job".  
  
  Argument type: Boolean |
| exclusiveMode | Use one of the following options:  
  - None - the "default", which does not retain a resource.  
  - Job - keeps the resource for the duration of the job step. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" is used in a job step. Future steps for this job will use this resource in preference to other resources--if this resource meets the needs of the job steps and its step limit is not exceeded.  
  - Step - keeps the resource for the duration of the job step.  
  - Call - keeps the resource for the duration of the procedure that called this job step, which is equivalent to 'job' for top level steps.  
  
  Argument type: ExclusiveMode |
| logFileName | A custom log file name produced by running the job step. By default, ElectricFlow assigns a unique name for this file.  
  
  Argument type: String |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| parallel     | If set, indicates this job step should run at the same time as adjacent job steps marked to run as parallel also. The value for parallel is a `<Boolean flag >0|1|true|false>`. Defaults to "false".  
  Argument type: Boolean |
| postProcessor| The name of a program to run after a job step completes. This program looks at the job step output to find errors and warnings. ElectricFlow includes a customizable program called "postp" for this purpose. The value for postProcessor is a command string for invoking a post-processor program in the platform shell for the resource (cmd for Windows, sh for UNIX).  
  Argument type: String |
### Arguments

<table>
<thead>
<tr>
<th>precondition</th>
</tr>
</thead>
</table>

### Descriptions

The `precondition` property (if it exists) is copied to the job step when the step is created. When the job step is eligible to transition from pending to runnable, the precondition is evaluated. If the precondition result is empty, `false`, or "0", the step remains in the pending state. Any other value allows the step to proceed to the runnable state.

**Note:** A precondition property allows steps to be created with "pause", which then pauses the procedure. In a paused state, all currently running steps continue, but no additional steps will start.

Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a precondition is evaluated.

A precondition is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a "0" or "false" is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.

**Argument type:** String

**Precondition example:**

Assume we defined these 4 steps:

1. Build object files and executables
2. Build installer
3. Run unit tests
4. Install bits on test system

Step 1 is an ordinary serial step.
Steps 2 and 3 can run in parallel because they depend only on step 1's completion.
Step 4 depends on step 2, but not step 3.

You can achieve optimal step execution order with preconditions:

- Make steps 2-4 run in parallel.
- Step 2 needs a job property set at the end of its step to indicate step 2 is completing
  `/myJob/buildInstallerCompleted=1`.
- **Set a precondition in step 4:**
  `[/myJob/buildInstallerCompleted]`
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| releaseExclusive | <Boolean flag - 0|1|true|false> Declares whether or not this job step will release its resource, which is currently held "exclusively".  
**Note:** Setting this flag to "true" is the same as setting releaseMode to release.  
Argument type: Boolean                                                                                                                                                                                                                                                                                                                                                       |
| releaseMode     | Use one of the following options:  
- **none** - the "default" - no action if the resource was not previously marked as "retain."  
- **release** - releases the resource at the end of this job step. If the resource for the job step was previously acquired with "Retain exclusive" (either by this job step or some preceding job step), the resource exclusivity is canceled at the end of this job step. The resource is released in the normal way so it may be acquired by other jobs.  
- **releaseToJob** - allows a job step to promote a "step exclusive" resource to a Job exclusive resource.  
Argument type: ReleaseMode                                                                                                                                                                                                                                                                                                                                                     |
| resourceName    | Name for the resource; must be unique among all resources.  
Argument type: String                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| shell           | Where shell is the name of a program used to execute commands contained in the "command" field. The name of a temporary file containing commands will be appended to the end of this invocation line. Normally, this file is a command shell, but it can be any other command line program. The default is to use the standard shell for the platform it runs on (cmd for Windows, sh for UNIX). Applicable to command steps only.  
Argument type: String                                                                                                                                                                                                                                                                                                                                                                                                               |
**status**

The *status* argument determines the "starting" job status. This is useful if you want to immediately go into a specific status without having to use `modifyJobStep` to update the status. Defaults to pending.

Possible values for *status*:

- **pending** – The job step is not yet runnable.
- **runnable** – The job step is ready to run.
- **scheduled** – The job step is scheduled to run.
- **running** – The job step is executing.

Argument type: `JobStatus`  

**subprocedure**

The name of the nested procedure to call when this job step runs. If a subprocedure is specified, do not include the `command` or `commandFile` options.

Argument type: `String`  

**subproject**

If a *subprocedure* argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.

Argument type: `String`  

**timelimit**

The maximum length of time the job step is allowed to run. After the time specified, the job step will be aborted. The time limit is specified in units that can be hours, minutes, or seconds.

Argument type: `String`  

**timelimitUnits**

Specify `hours | minutes | seconds` for time limit units.

Argument type: `TimeLimitUnits`  

**workingDirectory**

The `ElectricFlow` agent sets this directory as the "current working directory," when running the command contained in the job step. If no working directory is specified in the job step, `ElectricFlow` uses the directory it created for the job in the `ElectricFlow` workspace as the working directory.

**Note:** If running a job step on a proxy resource, this directory must exist on the proxy target.

Argument type: `String`  

**workspaceName**

The name of the workspace where this job step log files will be stored.

Argument type: `String`
Positional arguments

jobStepId

Response

Returns a modified jobStep object.

ec-perl

syntax: $cmdr->modifyJobStep (<jobStepId>, {<optional>});

Example

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

tool

syntax: ectool modifyJobStep <jobStepId> ...

Example

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

waitForJob

Waits until the specified job reaches a given status or the timeout expires.

This command works only with ec-perl.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>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</td>
</tr>
<tr>
<td>timeout</td>
<td>(Optional) The number of seconds to wait before giving up on a request. The default is 60 seconds. Argument type: Integer</td>
</tr>
<tr>
<td>finalStatus</td>
<td>(Optional) The status to wait for. Must be either &quot;running&quot; or &quot;completed&quot; (the default is &quot;completed&quot;). Argument type: String</td>
</tr>
</tbody>
</table>

Positional arguments

jobId
Response
Returns the result from the final getJobStatus query.

ec-perl

**syntax:**
```
$cmdr->waitForJob(<jobId>, <timeout>, <finalStatus>);
```

**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");
```

API Commands - Parameter Management

**attachParameter** on page 357
**createActualParameter** on page 359
**createFormalParameter** on page 362
**deleteActualParameter** on page 366
**deleteFormalParameter** on page 368
**detachParameter** on page 370
**getActualParameter** on page 372
**getActualParameters** on page 374
**getFormalParameter** on page 376
**getFormalParameterOptions** on page 378
**getFormalParameters** on page 380
**modifyActualParameter** on page 382
**modifyFormalParameter** on page 384
**validateFormalParameters** on page 387

**attachParameter**
Attaches a formal parameter to a step.
Attaching a parameter allows a step to use the credential (passed in a parameter) as an actual parameter to a subprocedure call or directly in a *getFullCredential* call in a command step. For more information about parameters, click here.
You must specify `projectName`, `procedureName`, `stepName`, and `formalParameterName`.

**Note:** The `attachParameter` command in ElectricFlow 6.0 is not backward compatible with previous ElectricFlow releases.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>procedureName</code></td>
<td>The name of the procedure to which the parameter is attached.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>stepName</code></td>
<td>The name of the procedure step to which the parameter is attached.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>formalParameterName</code></td>
<td>The name of the parameter with a credential reference.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>(Optional) The name of the application to which the parameter is attached.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>componentApplicationName</code></td>
<td>(Optional) The name of the component in the application to which the parameter is attached.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>componentName</code></td>
<td>(Optional) The name of the component to which the parameter is attached.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>gateType</code></td>
<td>(Optional) The type of the gate to which the parameter is attached: POST or PRE.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: GateType</td>
</tr>
<tr>
<td><code>pipelineName</code></td>
<td>(Optional) The name of the pipeline to which the parameter is attached.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>processName</code></td>
<td>(Optional) The name of the process to which the parameter is attached.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
**processStepName** | (Optional) The name of the process step to which the parameter is attached.  
Argument Type: String

**stageName** | (Optional) The name of the stage task to which the parameter is attached.  
Argument Type: String

**stateDefinitionName** | (Optional) The name of the workflow state definition to which the parameter is attached.  
Argument Type: String

**taskName** | (Optional) The name of the task to which the parameter is attached.  
Argument Type: String

**workflowDefinitionName** | (Optional) The name of the workflow definition which the parameter is attached when the parameter is attached to a workflow state definition.  
Argument Type: String

**Positional arguments**  
projectName, formalParameterName

**Response**  
None or status OK message.

**ec-perl**  
syntax: `$cmdr->attachParameter(<projectName>, <formalParameterName>, {<optionals>});`

**Example**  
```perl
$cmdr->attachParameter("Default", "SCM Credential");
```

**ectool**  
syntax: `ectool attachParameter <projectName> <formalParameterName> [optionals]

**Example**  
```bash
ectool attachParameter "Default" "Default "SCM Credential"
```

Back to Top

**createActualParameter**  
Creates a new actual parameter for a step that calls a nested procedure. The parameter is passed to the nested procedure when the step runs. At run time, the actual parameter name must match the name of a formal
parameter in the nested procedure.

**Passing Actual Parameters**

You can use actual parameters in three types of API calls:

- calling `runProcedure` to start a new job
- setting up a schedule
- creating or modifying a subprocedure step

For example, when you call `runProcedure` using `ectool`, set the actual parameters to the procedure on the command line using the optional argument `--actualParameter`, followed by a list of `name/value` pairs. The following is an example of calling a procedure named `MasterBuild`:

```bash
ectool runProcedure "project A" --procedureName "MasterBuild" --actualParameter Branch=main Type=Debug
```

To make this call using the Perl API, define a list. Each element of the list is an anonymous hash reference that specifies one of the actual parameters. Now you can pass a reference to the list as the value of the `actualParameter` argument.

Here is the same example called via the Perl API:

```perl
# Run the procedure
$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:

```xml
<createStep>
  <projectName>MyProject</projectName>
  <procedureName>MyProcedure</procedureName>
  <stepName>Step1</stepName>
  <actualParameter>
    <actualParameterName>parm1</actualParameterName>
    <value>myval</value>
  </actualParameter>
  <actualParameter>
    <actualParameterName>parm2</actualParameterName>
    <value>val2</value>
  </actualParameter>
</createStep>
```

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

```perl
{ ..., => ..., actualParameter => [{actualParameterName => 'parm1', value => 'myval'}, {actualParameterName => 'parm2', ...
```
In other words, the value of the `actualParameter` key in the optional arguments hash is a list of hashes, each representing one actual parameter. If the sub-procedure call takes only one actual parameter, the value of the `actualParameter` key can be specified as just the hash representing the one parameter:

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

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the procedure. The project name must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the step. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step that calls a subprocedure. Argument type: String</td>
</tr>
<tr>
<td>actualParameterName</td>
<td>The name of the parameter. This name must be unique within the step, and at run time, it must match the name of a formal parameter in the subprocedure. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application when the actual parameter is on an application process step. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component when the actual parameter is on a component process step. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the actual parameter is on a process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when if the actual parameter is on a process step. Argument type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release if the actual parameter is on a release. Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>value</td>
<td>(Optional) This value is passed to the subprocedure as the value of the matching formal parameter. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments
- projectName, procedureName, stepName, actualParameterName

### Response
None or status OK message.

### ec-perl
**Syntax:**
```
s/cmdr->createActualParameter(<projectName>, <procedureName>, <stepName>, <actualParameterName>, {<optionals>});
```

**Example**
```
s/cmdr->createActualParameter("Sample Project", "CallSub", "Step1", "Extra Parm", {value => "abcd efg"});
```

### ectool
**Syntax:**
```
ectool <projectName> <procedureName> <stepName> <actualParameterName> [optionals]
```

**Example**
```
ectool createActualParameter "Sample Project" "CallSub" "Step1" "Extra Parm" --value "abcd efg"
```

### createFormalParameter

Creates a new formal parameter.

You must specify projectName, procedureName, and formalParameterName.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the procedure. The project name must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the parameter. Argument type: String</td>
</tr>
<tr>
<td>formalParameterName</td>
<td>The name for the formal parameter that is used when the procedure is invoked to specify a value for the parameter. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application when the actual parameter is on an application process step. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component when the actual parameter is on a component process step. Argument type: String</td>
</tr>
<tr>
<td>defaultValue</td>
<td>(Optional) This value is used for the formal parameter when a value is not provided and the procedure is invoked. Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: `&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt; Argument type: String</td>
</tr>
<tr>
<td>expansionDeferred</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>label</td>
<td>(Optional) The display label. Argument type: String</td>
</tr>
<tr>
<td>orderIndex</td>
<td>(Optional) The display order index starting at 1. Argument type: Integer</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Application type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the formal parameter is on a process. Application type: String</td>
</tr>
<tr>
<td>required</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Application type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of a workflow state. Application type: String</td>
</tr>
<tr>
<td>type</td>
<td>(Optional) The <code>type</code> can be any string value. This argument is used primarily by the web interface to represent custom form elements. When the type is the <code>credential</code> string value, the server expect a credential as the parameter value. Application type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Application type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of a workflow. Application type: String</td>
</tr>
</tbody>
</table>

### 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

None or status OK message.

### 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/elec_customEditorData/parameters/CheckoutSources/checkedValue", "true");
    $ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/elec_customEditorData/parameters/CheckoutSources/uncheckedValue", "false");
    $ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/elec_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/elec_customEditorData/parameters/BuildType/options/optionCount", "2");
    $ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/elec_customEditorData/parameters/BuildType/options/type", "list");
    $ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/elec_customEditorData/parameters/BuildType/options/option1/text", "one");
    $ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/elec_customEditorData/parameters/BuildType/options/option1/value", "1");
    $ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/elec_customEditorData/parameters/BuildType/options/option2/text", "two");
    $ec->setProperty("/projects/$newProjectName/procedures/$buildprocedurename/elec_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.

**deleteActualParameter**

Deletes an actual parameter.

You must specify a `projectName`, `procedureName`, `stepName`, and `actualParameterName`. 
Arguments | Descriptions
--- | ---
projectName | The name of the project that contains this actual parameter. The name must be unique among all projects. Argument type: String
procedureName | The name of the procedure that contains the step with this parameter. Argument type: String
stepName | The name of the step that contains the actual parameter. Argument type: String
actualParameterName | The name of the actual parameter to delete. Argument type: String
applicationName | (Optional) The name of the application when the actual parameter is on an application process step. Argument type: String
componentName | (Optional) The name of the component when the actual parameter is on a component process step. Argument type: String
processName | (Optional) The name of the process when the actual parameter is on a process step. Argument type: String
processStepName | (Optional) The name of the process step when the actual parameter is on a process step. Argument type: String
releaseName | (Optional) The name of the release if the actual parameter is on a release. Argument type: String
scheduleName | (Optional) The name of the schedule containing the actual parameter. Argument type: String
stateDefinitionName | (Optional) The name of the state definition. Argument type: String
transitionDefinitionName | (Optional) The name of the transition definition. Argument type: String
Arguments | Descriptions
---|---
workflowDefinitionName | (Optional) The name of the workflow definition. Argument type: String

**Positional arguments**
projectName, procedureName, stepName, actualParameterName

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

**ec-perl**
syntax: $cmdr->deleteActualParameter(<projectName>, <procedureName>, <stepName>, <actualParameterName>, {<optionals>});

Example
$cmdr->deleteActualParameter('Sample Project', 'CallSub', 'Step1', 'Different Param');

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

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

**deleteFormalParameter**
Deletes a formal parameter.

You must specify projectName, procedureName, and formalParameterName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project that contains the procedure or parameter to delete. The name must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure that contains this parameter. Argument type: String</td>
</tr>
<tr>
<td>formalParameterName</td>
<td>(Optional) The name of the formal parameter to delete. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application when the formal parameter is on an application process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component when the formal parameter is on a component process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the formal parameter is on a process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the workflow state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of a workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**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>});`

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

*Syntax:* `$cmdr->deleteFormalParameter(<projectName>, <procedureName>, <formalParameterName>, {<optionals>});`
**Example**

```bash
$cmdr->deleteFormalParameter("Sample Project", "Build Name");
```

**ectool**

**syntax:** `ectool deleteFormalParameter <projectName> <formalParameterName> [optionals]`

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

**syntax:** `ectool deleteFormalParameter <projectName> <procedureName> <formalParameterName> [optionals]`

**Example**

```bash
ectool deleteFormalParameter "Sample Project" "Build Name"
```

**detachParameter**

Detaches a formal parameter from a step.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project that contains this parameter. The name must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>procedureName</code></td>
<td>The name of the procedure that contains this parameter.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>stepName</code></td>
<td>The name of the step where this parameter is currently attached.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>formalParameterName</code></td>
<td>The name of the formal parameter to detach.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>(Optional) The name of the application.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>componentApplicationName</code></td>
<td>(Optional) The name of the component in the application.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>componentName</code></td>
<td>(Optional) The name of the component.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gateType</td>
<td>(Optional) The type of the gate. Argument type: GateType</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process that contains this parameter. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
</tbody>
</table>

### 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 "Test Proj" "Run Build" "Get Sources" "SCM Credential"
```
getActualParameter

Retrieves an actual parameter by its name. For more information about parameters, click here.

You must specify an actualParameterName. If you need actual parameters on a step, the projectName, procedureName, and stepName arguments must be used together to specify the step.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameterName</td>
<td>The name of the actual parameter.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationEntityRevisionId</td>
<td>(Optional) The revision ID of the versioned object.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application when the actual parameter is on an application process step. The name must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component when the actual parameter is on a component process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>(Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created.</td>
</tr>
<tr>
<td></td>
<td>The system also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created.</td>
</tr>
<tr>
<td></td>
<td>You enter this argument to query a subprocedure call to the job step's parameter.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure when the actual parameter is on a procedure step.</td>
</tr>
<tr>
<td></td>
<td>If you need actual parameters on a step, the projectName, procedureName, and stepName arguments must be used together to specify the step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the actual parameter is on a process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the actual parameter is on a process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project to query for the parameter on a schedule or procedure step.</td>
</tr>
<tr>
<td></td>
<td>If you need actual parameters on a step, the projectName, procedureName, and stepName arguments must be used together to specify the step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release if the actual parameter is on a release.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule to query for the parameter on a schedule.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the workflow state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the workflow state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step to query for the parameter on the step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the workflow transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the workflow transition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
Positional arguments

actualParameterName

Response

One actualParameter element.

ec-perl

Syntax: $cmdr->getActualParameter(<actualParameterName>, [optionals]);

Example

$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"

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 projectsName, procedureName, and stepName to specify the step.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationEntityRevisionId</td>
<td>(Optional) The revision ID of the versioned object.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application when the actual parameters are on an application process step. The name must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component when the actual parameters are on a component process step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobId</td>
<td>(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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>procedureName</td>
<td>(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</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the actual parameters are on a process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the actual parameters are on a process step. Argument type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release if the actual parameter is on a release. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(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</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule containing parameters. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the workflow state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the workflow state. Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th></th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step containing parameters.</td>
</tr>
<tr>
<td></td>
<td>To find parameters on a step, you must use projectName, procedureName, and stepName to specify the step.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transi_tionDefinitionName</td>
<td>(Optional) The name of the workflow transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the workflow transition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

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

**Response**
Zero or more actualParameter elements.

**ec-perl**

*Syntax:* $cmdr->getActualParameters ([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"
```

### getFormalParameter
Retrieves a formal parameter by its name.

You must specify projectName and formalParameterName.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the procedure. The name must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>formalParameterName</td>
<td>The name for the formal parameter that is used when the procedure is invoked to specify a value for the parameter. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application when the formal parameter is on an application process step. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component when the formal parameter is on a component process step. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure containing the formal parameter. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the formal parameter is on a process step. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the workflow state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the workflow state. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow. Argument type: String</td>
</tr>
</tbody>
</table>

**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:**
```perl
$cmdr->getFormalParameter(<projectName>, <formalParameterName>, {optionals});
```

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

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

**Example**

```
$cmdr->getFormalParameter("Test", "Get Sources");
```

**ectool**

**syntax:**
```bash
ectool getFormalParameter <projectName> <formalParameterName> [optionals]
```

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

**syntax:**
```bash
ectool getFormalParameter <projectName> <procedureName> <formalParameterName> [optionals]
```

**Example**

```
ectool getFormalParameter Test "Get Sources"
```

**getFormalParameterOptions**

Gets 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 **procedureName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>formalParameterName</td>
<td>Name of the parameter for which the options are requested. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>Name of the procedure. Argument type: String</td>
</tr>
<tr>
<td>actualParameters</td>
<td>(Optional) Parameters passed as arguments to the script. Argument type: String</td>
</tr>
<tr>
<td>configurationParameters&lt;</td>
<td>(Optional) Configuration parameter values passed as arguments to the script. Argument type: Map</td>
</tr>
<tr>
<td>credentials</td>
<td>(Optional) Credentials to be used in the script. Argument type: Collection</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginName</td>
<td>(Optional) Name of a plugin.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) Name of a project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

`formalParameterName, procedureName`

### Response

One `actualParameter` element.

### ec-perl

**syntax:**

```perl
$cmdr->getActualParameter(<actualParameterName>, {optionals});
```

**Example**

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

### ectool

**syntax:**

```bash
ectool getActualParameter <actualParameterName> [optionals]
```

**Example**

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

Back to Top

### Required Arguments

`formalParameterName`

**Description:** Name of the parameter for which the options are requested.

**Argument Type:** String

`procedureName`

**Description:** The name of a procedure.

**Argument Type:** String

### Optional Arguments

`actualParameters`

**Description:** Parameters passed as arguments to the script.

**Argument Type:** Map
configurationParameters

Description: Configuration parameter values passed as arguments to the script.
Argument Type: Map

credentials

Description: Credentials to be used in the script
Argument Type: Collection

pluginName

Description: The name of a plugin.
Argument Type: String

projectName

Description: The name of a project.
Argument Type: String

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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the parameters. The name must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationEntityRevisionId</td>
<td>(Optional) The revision ID of the versioned object. Argument type: UUID</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application when the formal parameters are on an application process step. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component when the formal parameters are on a component process step. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure when the formal parameters are on a procedure. When using this argument, you must also enter projectName. Argument type: String</td>
</tr>
</tbody>
</table>
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the formal parameters are on a process step. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule. When using this argument, you must also enter <code>projectName</code>. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the workflow state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the workflow state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step. When using this argument, you must also enter <code>projectName</code> and <code>procedureName</code>. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow. Argument type: String</td>
</tr>
</tbody>
</table>

### 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("Test", {procedureName => "Build");
```

#### ectool

**syntax:**

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

**Example**

```
getFormalParameters Test --procedureName Build
```

[Back to Top](#)
modifyActualParameter

Modifies an existing actual parameter. An actual parameter is a name/value pair passed to a subprocedure. This command supports renaming the actual parameter and setting its value. For more information about parameters, click here.

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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing this parameter. The name must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the step with this parameter. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step containing this parameter. Argument type: String</td>
</tr>
<tr>
<td>actualParameterName</td>
<td>The name of the actual parameter to modify. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application when the actual parameters are on an application process step. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component when the actual parameters are on a component process step. Argument type: String</td>
</tr>
<tr>
<td>newName</td>
<td>(Optional) New name of the parameter. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the actual parameters are on a process step. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the actual parameters are on a process step. Argument type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release if the actual parameter is on a release. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the workflow state definition. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the workflow transition definition. Argument type: String</td>
</tr>
<tr>
<td>value</td>
<td>(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</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
</tbody>
</table>

**Response**

None or a status OK message.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* ectool modifyActualParameter <projectName> <procedureName> <stepName> <actualParameterName> [optionals]

*Example*

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing this parameter. The name must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>formalParameterName</td>
<td>The name for this formal parameter. It is used when the procedure is invoked to specify a value for the parameter. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application when the formal parameters are on an application process step. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component when the formal parameters are on a component process step. Argument type: String</td>
</tr>
<tr>
<td>defaultValue</td>
<td>This value is used for the formal parameter if one is not provided when the procedure is invoked. Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt; Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>expansionDeferred</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>label</td>
<td>The display label. Argument type: String</td>
</tr>
<tr>
<td>newName</td>
<td>New name for the parameter. Argument type: String</td>
</tr>
<tr>
<td>orderIndex</td>
<td>The display order index starting at 1. Argument type: Integer</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure containing this parameter. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the formal parameter is on a process step. Argument type: String</td>
</tr>
<tr>
<td>required</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of a workflow state. Argument type: String</td>
</tr>
<tr>
<td>type</td>
<td>(Optional) The <code>type</code> can be any string value. This argument is used primarily by the web interface to represent custom form elements. When the type is the <code>credential</code> string value, the server expect a credential as the parameter value. Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of a workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

### 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

None or a status OK message.

### `ec-perl`

For procedural parameters in **ElectricFlow 5.0 and later**:

**syntax:**

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

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

**syntax:**

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

**Example**

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

### `ectool`

For procedural parameters in **ElectricFlow 5.0 and later**:

**syntax:**

```
ectool modifyFormalParameter <projectName> <formalParameterName> [optionals]
```

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>
```

**Example**

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

Validates input parameters for a procedure using the validation script registered for it.
You must enter the `procedureName` argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>procedureName</code></td>
<td>The name of a procedure. Argument type: String</td>
</tr>
<tr>
<td><code>actualParameters</code></td>
<td>(Optional) Parameters passed as arguments to the script. Argument type: String</td>
</tr>
<tr>
<td><code>configurationParameters</code></td>
<td>(Optional) Configuration parameter values passed as arguments to the script. Argument type: Map</td>
</tr>
<tr>
<td><code>credentials</code></td>
<td>(Optional) Credentials to be used in the script. Argument type: Collection</td>
</tr>
<tr>
<td><code>pluginName</code></td>
<td>(Optional) The name of a plugin. Argument type: String</td>
</tr>
<tr>
<td><code>projectName</code></td>
<td>(Optional) The name of a project. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`

**Response**

None or a status OK message.

**ec-perl**

`syntax:` $cmdr-&gt;validateFormalParameters(<procedureName>, {<optionals>});

**Example**

$cmdr-&gt;validateFormalParameters("Shopping Cart", {projectName =&gt; "Default"});

**ectool**

`syntax:` ectool validateFormalParameters <procedureName> [optionals]

**Example**

validateFormalParameters "Shopping Cart" --projectName "Default"

Back to Top
API Commands - Pipelines and Releases

abortPipelineRun on page 389  
completeRelease on page 390  
createDeployer on page 391  
cREATEDEPLOYERAPPLICATION on page 391  
createDeployerConfiguration on page 393  
cREATENOTE on page 394  
createpipeline on page 395  
createRelease on page 396  
createStage on page 397  
createTask on page 398  
deleteNote on page 402  
deletesPipeline on page 403  
deletePipelineRun on page 403  
deleteRelease on page 404  
deleteStage on page 405  
deleteTask on page 406  
getDeployerApplication on page 407  
getDeployerConfiguration on page 408  
getDeployerConfigurations on page 409  
getNote on page 410  
getNotes on page 411  
getPipeline on page 412  
getPipelineRuntimeDetails on page 413  
getPipelineRuntimes on page 413  
getPipelineStageRuntimeDeployerTasks on page 415  
getPipelineStageRuntimeTasks on page 416  
getPipelines on page 417  
getRelease on page 418  
getReleaseInventory on page 419  
getReleases on page 420  
getStage on page 422  
getStages on page 423  
getTask on page 424  
getTasks on page 425
modifyDeployer on page 426
modifyDeployerApplication on page 426
modifyDeployerConfiguration on page 428
createNote on page 394
modifyPipeline on page 430
modifyRelease on page 431
modifyStage on page 433
modifyTask on page 434
removeDeployerApplication on page 438
removeDeployerConfiguration on page 439
runPipeline on page 440
startRelease on page 441
validateDeployer on page 442

abortPipelineRun

Aborts a pipeline run.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowRuntimeId</td>
<td>(Optional) The ID of the flow runtime.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: UUID</td>
</tr>
<tr>
<td>flowRuntimeName</td>
<td>(Optional) The name of the flow runtime.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>force</td>
<td>(Optional) &lt;Boolean flag&gt; - 0</td>
</tr>
<tr>
<td></td>
<td>When this argument is set to true or 1, the running tasks are force-aborted.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Boolean</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Response

None or a status OK message.

ec-perl

syntax: $<object>->abortPipelineRun( {<optionals>});
**Example**

```bash
$ec->abortPipelineRun({flowRuntimeName => "Final Release", projectName => "Production"});
```

**ectool**

**syntax:** ectool abortPipelineRun [optionals...]

**Example**

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

**Back to Top**

---

**completeRelease**

Completes release operation.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Map</td>
</tr>
<tr>
<td>releaseId</td>
<td>(Optional) The release ID.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**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 for a project or a release.

You must specify the projectName and the deployerName arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>deployerName</td>
<td>Name of the Deployer that must be unique within the project.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Positional arguments

projectName, deployerName

Response

None or a status OK message.

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"

createDeployerApplication

Adds a Deployer application to the specified release.

You must specify the projectName and applicationName arguments.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>Name of the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationProjectName</td>
<td>(Optional) The name of the project containing the application. If this argument is not specified, the default is the release project name.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>orderIndex</td>
<td>(Optional) The order in which the applications are deployed, starting from 1.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the application process.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

projectName, applicationName

### Response

None or a status OK message.

**ec-perl**

**syntax:**

```
$mailobject->createDeployerApplication($projectName, $applicationName, $optionals);
```

**Example**

```perl
$mailobject->createDeployerApplication("Default", "Shopping Cart", $optionals);
```

**ectool**

**syntax:**

```
ectool createDeployerApplication <projectName> <applicationName> [optionals...]
```

**Example**

```bash
ectool createDeployerApplication "Default" "Shopping Cart" --applicationProjectName "myApplications"
```
createDeployerConfiguration

Adds a Deployer configuration to the Deployer application.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>Name of the stage of a pipeline attached to a release if specified.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>actualParameters</td>
<td>(Optional) Actual parameters.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Map</td>
</tr>
<tr>
<td>applicationProjectName</td>
<td>(Optional) The name of the project containing the application. If this</td>
</tr>
<tr>
<td></td>
<td>argument is not specified, the default is the release project name.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) Name of the environment.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentProjectName</td>
<td>(Optional) Name of the project containing the specified environment or</td>
</tr>
<tr>
<td></td>
<td>environment template. If this argument is not specified, the default is</td>
</tr>
<tr>
<td></td>
<td>the release project name.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>skipDeploy</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>When this argument is set to <code>true</code> or <code>1</code>, the application is not</td>
</tr>
<tr>
<td></td>
<td>deployed to an environment.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, applicationName, stageName`
**Response**

None or a status OK message.

**ec-perl**

*Syntax:* $<object>-createDeployerConfiguration(<projectName>, <applicationName>, <stageName>, {<optionals>});

*Example*

$ec->createDeployerConfiguration("Default", "Shopping Cart", "Publish components", {applicationProjectName => "myApplications"});

**ectool**

*Syntax:* ectool createDeployerConfiguration <projectName> <applicationName> <stageName> [optionals...]

*Example*

ectool createDeployerConfiguration "Default" "Shopping Cart" "Publish components" --applicationProjectName "myApplications"

**createNote**

Creates a new note.

You must specify the *note* argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>note</td>
<td>Notes about the entity.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>noteName</td>
<td>(Optional) Name of the note.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) Project name of the entity that owns the note.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

*note*

**Response**

None or a status OK message.
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 the **pipelineName** arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>Name of the pipeline that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>enabled</td>
<td>(Optional) <strong>&lt;Boolean flag&gt;</strong> - 0</td>
</tr>
<tr>
<td></td>
<td>When this argument is set to true or 1, the pipeline is enabled.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>type</td>
<td>(Optional) Type of pipeline.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: PipelineType</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, pipelineName

**Response**

None or a status OK message.
**createRelease**

Creates a new release for a project.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>Name of the release. Argument Type: String</td>
</tr>
<tr>
<td>actualParameters</td>
<td>(Optional) Actual parameters. Argument Type: Map</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline. Argument Type: String</td>
</tr>
<tr>
<td>pipelineProjectName</td>
<td>(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</td>
</tr>
</tbody>
</table>
createRelease

Creates a release for a project.

You must specify the projectName and the releaseName arguments.

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

Argument types:

- plannedEndDate: (Optional) The date when this release is expected to end (for example, 2006-05-15).
- Argument Type: String

- plannedStartDate: (Optional) The date when this release is expected to begin (for example, 2006-05-15).
- Argument Type: String

Response

None or a status OK message.

**ec-perl**

**syntax:**

```perl
$<object>-&gt;createRelease(<projectName>, <releaseName>, {<optionals>});
```

**Example**

```perl
$ec-&gt;createRelease("Default", "Production", {description =&gt; "Pet Shop Web Site", pipelineName =&gt; "Daily Update"});
```

**ectool**

**syntax:**

```bash
ectool createRelease <projectName> <releaseName> [optionals...]
```

**Example**

```bash
ectool createRelease "Default" "Production" --description "Pet Shop Web Site" --pipelineName "Daily Update"
```

Back to Top
## createStage

**Purpose**

Creates a new stage in a pipeline.

**Argument Descriptions**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>stageName</td>
<td>Name of the stage.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>afterStage</td>
<td>(Optional) The stage that is placed after the new stage.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>beforeStage</td>
<td>(Optional) The stage that is placed before the new stage.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline to which the stage belongs.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, stageName`

**Response**

None or a status OK message.

**Examples**

**ec-perl**

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

*Example*

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

**ectool**

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

*Example*

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

**createTask**

Creates a new task for a task container.

You must specify the `projectName` and `taskId` arguments.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>taskName</td>
<td>Name of the task. Argument Type: String</td>
</tr>
<tr>
<td>actualParameters</td>
<td>(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called process. Argument Type: Map</td>
</tr>
<tr>
<td>afterTask</td>
<td>(Optional) The task that is placed after the new task. Argument Type: String</td>
</tr>
<tr>
<td>approvers</td>
<td>(Optional) A list of task approvers who receive the notification. Argument Type: Collection</td>
</tr>
<tr>
<td>beforeTask</td>
<td>(Optional) The task that is placed before the new task. Argument Type: String</td>
</tr>
<tr>
<td>credentials</td>
<td>(Optional) The credentials to be used in the task. Argument type: Collection</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String</td>
</tr>
<tr>
<td>enabled</td>
<td>(Optional) <code>&lt;Boolean flag</code> - 0</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) Name of the environment to create from a template. Argument Type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) Name of the environment template to use. Argument Type: String</td>
</tr>
<tr>
<td>errorHandling</td>
<td>(Optional) Type of error handling for this task. Argument Type: ErrorHandling</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) The type of the gate. Argument Type: GateType</td>
</tr>
<tr>
<td>instruction</td>
<td>(Optional) Instruction associated with the task. Argument Type: String</td>
</tr>
</tbody>
</table>
| keepOnError   | (Optional) `<Boolean flag - 0|1|true|false>`  
When this argument is set to true or 1, the system is set to keep environment if an error occurs. The default is false or 0.  
Argument type: Boolean |
| notificationTemplate | (Optional) String containing email formatting instructions for generating notifications.  
Argument type: String |
| pipelineName  | (Optional) Name of the pipeline to which the stage belongs. Argument Type: String                                                            |
| skippable     | (Optional) `<Boolean flag - 0|1|true|false>`  
When this argument is set to true or 1, the task can be skipped in the pipeline.  
Argument type: Boolean |
| snapshotName  | (Optional) Name of the snapshot associated with the application. Argument Type: String                                                       |
| stageName     | (Optional) Name of the stage to which the task belongs. Argument Type: String                                                               |
| startTime     | (Optional) The time to begin invoking this task. The time is formatted hh:mm using the 24-hours clock (for example, 17:00). Argument Type: String |
| subapplication| (Optional) Name of the application that owns the subprocess. Argument Type: String                                                            |
| subpluginKey  | (Optional) Name of the pluginKey for a procedure when the procedure is referenced. Argument Type: String                                         |
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>subprocedure</td>
<td>(Optional) Name of the subprocedure when a procedure is referenced.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>subprocess</td>
<td>(Optional) Name of the ElectricFlow process.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>subproject</td>
<td>(Optional) Name of the project in which the procedure runs.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>subworkflowDefinition</td>
<td>(Optional) Name of the workflow definition in which the workflow definition is referenced.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>subworkflowStartingState</td>
<td>(Optional) Name of the starting state in the specified workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>taskProcessType</td>
<td>(Optional) The type of the process that the task can invoke.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: TaskProcessType</td>
</tr>
<tr>
<td>taskType</td>
<td>(Optional) The type of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: TaskType</td>
</tr>
<tr>
<td>tierResourceCounts</td>
<td>(Optional) The resource count per resource template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Map</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

projectName, taskName

### Response

None or a status OK message.

### ec-perl

**syntax:**

```perl
$<object>->createTask(<projectName>, <taskName>, {<optionals>});
```

**Example**

```perl
$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"`

**deleteNote**

Deletes a note associated with an entity.

You must specify the `note` argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>note</td>
<td>Notes provided about the entity.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) Project name of the entity that owns the note.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**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.

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

**Positional arguments**

**projectName, pipelineName**

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* $<object>->deletePipeline(<projectName>, <pipelineName>);

*Example*

$ec->createPipeline("Default", "Web Server Image");

**ectool**

*Syntax:* ectool deletePipeline <projectName> <pipelineName>

*Example*

ectool createPipeline "Default" "Web Server Image"

**deletePipelineRun**

Deletes a pipeline runtime.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowRuntimeId</td>
<td>(Optional) The ID of the flow runtime. Argument Type: UUID</td>
</tr>
</tbody>
</table>
### deletePipelineRun

**Deleted a release.**

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

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

**Positional arguments**

`projectName, releaseName`

**Response**

None or a status OK message.
**deleteStage**

Deletes a stage in a project.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projectName</strong></td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><strong>stageName</strong></td>
<td>Name of the stage.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><strong>pipelineName</strong></td>
<td>(Optional) Name of the pipeline to which the stage belongs.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `stageName`

**Response**

None or a status OK message.

**ec-perl**

**syntax:**

```
$<object>->deleteStage(<projectName>, <stageName>, [<optionals>]);
```

**Example**

```
$ec->deleteStage("Default", "Preflight");
```

**ectool**

**syntax:**

```
ectool deleteStage <projectName> <stageName> [optionals...]
```

**Example**

```
ectool deleteRelease "Default" "Pet Store"
```
**Example**

dectool deleteStage "Default" "Preflight"

```
Example
detectTask
```

**deleteTask**

Deletes a task in a task container.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>taskName</td>
<td>Name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) The type of the gate.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: GateType</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline to which the stage belongs.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) Name of the stage to which the task belongs.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, taskName`

**Response**

None or a status OK message.

**ec-perl**

```
syntax: $<object>-&gt;deleteTask(<projectName>, <taskName>, {<optionals>});
```

**Example**

```
$ec-&gt;deleteTask("Default", "Save image", {pipelineName =&gt; "Beta build"});
```

**ectool**

```
syntax: ectool deleteTask &lt;projectName&gt; &lt;taskName&gt; [optionals...]
```

**Example**

```
ectool deleteTask "Default" "Save image" --pipelineName "Beta build"
```

Back to Top
**getDeployerApplication**

Gets the application in the release by name.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationProjectName</td>
<td>(Optional) The name of the project containing specified application. If not specified, it is defaulted to the release project name.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) Name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, applicationName`

**Response**

Gets a Deployer application and its details.

**ec-perl**

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

**Example**

```perl
$ec->getDeployerApplication("Default", "Verify versions", {applicationProjectName = "myApps"});
```

**ectool**

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

**Example**

```bash
ectool getDeployerApplication "Default" "Verify versions" --applicationProjectName "myApps"
```

**getDeployerApplications**

Gets all the applications in the release.
You must specify the `projectName` argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`

**Response**

Returns all the Deployer applications and their details.

**ec-perl**

*Syntax:*

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

*Example*

```
$ec-getDeployerApplications("Default", {releaseName => "Weekly Build"});
```

**ectool**

*Syntax:*

```
ectool getDeployerApplications <projectName> <applicationName> [optionals...]
```

*Example*

```
ectool getDeployerApplications "Default" --releaseName "Weekly Build"
```

**getDeployerConfiguration**

Gets a Deployer configuration.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>Name of the stage of a pipeline attached to a release if specified.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>
**ElectricFlow Perl API Commands**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationProjectName</td>
<td>(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</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release. Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, applicationName, stageName

**Response**

Returns the Deployer configuration and its details.

**ec-perl**

`syntax:`

```
$<object>-gtDeployerConfiguration(<projectName>, <applicationName>, <stageName>, {<optionals>});
```

**Example**

```perl
$ec->getDeployerConfiguration("Default", "Shopping Cart", "Publish components", {applicationProjectName => "myApplications"});
```

**ectool**

`syntax:`

```
ectool getDeployerConfiguration <projectName> <applicationName> <stageName> [optionals...]
```

**Example**

```ectool
ectool getDeployerConfiguration "Default" "Shopping Cart" "Publish components" --applicationProjectName "myApplications"
```

**getDeployerConfigurations**

Gets all the configurations in the Deployer.

You must specify the `projectName` argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) Name of the application. Argument Type: String</td>
</tr>
<tr>
<td>applicationProjectName</td>
<td>(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</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release. Argument Type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) Name of the stage of a pipeline attached to a release if specified. Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`

**Response**

Returns the list of Deployer configurations and their details.

**ec-perl**

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

*Example*

$ec->getDeployerConfigurations("Default", {applicationProjectName => "myApplication s"});

**ectool**

*Syntax:* `ectool getDeployerConfigurations <projectName> [optionals...]`

*Example*

ectool getDeployerConfigurations "Default" --applicationProjectName "myApplication s"

**getNote**

Gets a note associated with an entity.

You must specify the `noteName` argument.
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>noteName</td>
<td>Name of the note.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) Project name of the entity that owns the note.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- **noteName**

**Response**

Returns the selected note.

**ec-perl**

**syntax:**

```perl
$<object>-getNote(<noteName>, {<optionals>});
```

**Example**

```perl
$ec->getNote("Final Approval", {releaseName => "Production"});
```

**ectool**

**syntax:**

```bash
ectool getNote <noteName> [optionals...]
```

**Example**

```bash
ectool getNote "Final Approval" --releaseName "Production"
```

### getNotes

Gets all the notes associated with an entity.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>(Optional) Project name of the entity that owns the note.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**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.

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

**Positional arguments**

*projectName, pipelineName*

**Response**

Returns the specified pipeline element.

**ec-perl**

*Syntax:* $<object>->getPipeline(<projectName>, <pipelineName>);

*Example*

$ec->getPipeline("Default", "Beta test");

**ectool**

*Syntax:* ectool getPipeline <projectName> <pipelineName>
**Example**

ectool getPipeline "Default" "Beta test"

**getPipelineRuntimeDetails**

Retrieves pipeline runtime details.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowRuntimeIds</td>
<td>(Optional) List of the pipeline flowRunTime IDs.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Collection</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Returns the runtime details of the flow.

**ec-perl**

*syntax:* $<object>->getFlowRuntimeDetails({<optionals>});

**Example**

$ec->getFlowRuntimeDetails({flowRuntimeName => "First build", projectName => "Default"});

**ectool**

*syntax:* ectool getFlowRuntimeDetails [optionals...]

**Example**

ectool getFlowRuntimeDetails --flowRuntimeName "First build" --projectName "Default"

**getPipelineRuntimes**

Retrieves pipeline runs.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>firstResult</td>
<td>(Optional) First row of the results to return, based on how the results are paginated.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Integer</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>flowRuntimeId</td>
<td>(Optional) ID of the flow runtime. Argument Type: UUID</td>
</tr>
<tr>
<td>maxResults</td>
<td>(Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline. Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. (Optional) Argument Type: String</td>
</tr>
<tr>
<td>releaseId</td>
<td>(Optional) ID of the release. Argument Type: String</td>
</tr>
<tr>
<td>sortKey</td>
<td>(Optional) How to sort the results. Argument Type: String</td>
</tr>
<tr>
<td>sortOrder</td>
<td>(Optional) The order in which the results are sorted. Argument Type: SortOrder</td>
</tr>
<tr>
<td>statusExcludes</td>
<td>(Optional) List of statuses to exclude from the response. Argument Type: Collection</td>
</tr>
</tbody>
</table>

**Positional arguments**
None

**Response**
Returns pipeline runs.

**ec-perl**

`syntax:`

$<object>->getPipelineRuntimes({<optionals>});

**Example**

$ec->getPipelineRuntimes({pipelineName => "Beta test", projectName => "Default"});

**ectool**

`syntax:`

ectool getPipelineRuntimes [optionals...]

**Example**

ectool getPipelineRuntimes --pipelineName "Beta test" --projectName "Default"
getPipelineStageRuntimeDeployerTasks

Returns a list of pipeline stage Deployer Tasks and their details to display in Pipeline Run Details page.

You must specify the flowRuntimeId argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>flowRuntimeId</td>
<td>ID of the flow runtime. Argument Type: UUID</td>
</tr>
<tr>
<td>firstResult</td>
<td>(Optional) First row of the results to return, based on how the results are paginated. Argument Type: Integer</td>
</tr>
<tr>
<td>maxResults</td>
<td>(Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer</td>
</tr>
<tr>
<td>sortKey</td>
<td>How to sort the results. Argument Type: String</td>
</tr>
<tr>
<td>sortOrder</td>
<td>The order in which the results are sorted. Argument Type: SortOrder</td>
</tr>
<tr>
<td>stageName</td>
<td>Name of the stage. Argument Type: String</td>
</tr>
</tbody>
</table>

Positional arguments
flowRuntimeId

Response
Returns a list of Deployer Tasks for a pipeline stage and the details about them.

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

**Required Arguments**

flowRuntimeId  
*Description*: The id of the flow runtime  
*Argument Type*: UUID

stageName  
*Description*: The name of the stage  
*Argument Type*: String

taskId  
*Description*: The name of the Deployer Task  
*Argument Type*: String

**Optional Arguments**

firstResult  
*Description*: Result pagination: the first row to return.  
*Argument Type*: Integer

maxResults  
*Description*: Result pagination: the number of rows to return.  
*Argument Type*: Integer

---

**getPipelineStageRuntimeTasks**

Retrieves a list of pipeline stage tasks and the details about them that are displayed in the pipeline run view.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| flowRuntimeId | ID of the flow runtime.  
Argument Type: UUID |
| stageName    | Name of the stage.  
Argument Type: String |
| taskId       | Name of the Deployer Task.  
Argument Type: String |
Arguments | Descriptions
---|---
firstResult | (Optional) First row of the results to return, based on how the results are paginated. Argument Type: Integer
maxResults | (Optional) The number of rows to return, based on how the results are paginated. Argument Type: Integer
sortKey | (Optional) How to sort the results. Argument Type: String
sortOrder | (Optional) The order in which the results are sorted. Argument Type: SortOrder

Positional arguments
flowRuntimeId

Response
Returns a list of pipeline stage tasks and the details about them in the pipeline run view.

ec-perl

* syntax:*

```perl
<object>->getPipelineStageRuntimeDeployerTasks (<flowRuntimeId>, <stageName>, <taskName>, {<optionals>});
```

* Example *

```perl
$ec->getPipelineStageRuntimeDeployerTasks(4fa765dd-73f1-11e3-b67e-b0a420524165, "Find files", "Login to app", {firstResult => 2, maxResults => 200});
```

ectool

* syntax:*

```bash
ectool getPipelineStageRuntimeDeployerTasks <flowRuntimeId> <stageName> <taskName> [optionals...]
```

* Example *

```bash
ectool getPipelineStageRuntimeDeployerTasks 4fa765dd-73f1-11e3-b67e-b0a420524165 "Find files" "Login to app" --firstResult 2 --maxResults 200
```

getPipelines

Retrieves all the pipelines.
You must specify the `projectName` argument.
Arguments | Descriptions
--- | ---
**projectName** | Name of the project that must be unique among all projects. Argument Type: String

**Positional arguments**

**projectName**

**Response**

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"

**getRelease**

Retrieves a release by name.

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

Arguments | Descriptions
--- | ---
**projectName** | Name of the project that must be unique among all projects. Argument Type: String
**releaseName** | Name of the release. Argument Type: String

**Positional arguments**

**projectName, releaseName**

**Response**

Returns 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"

**getReleaseInventory**

Gets inventory artifacts created in a release.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td>releaseName</td>
<td>Name of the release.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, releaseName

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* $<object>-getReleaseInventory(<projectName>, <releaseName>);

*Example*

$ec->getReleaseInventory("Default", "Production");

**ectool**

*Syntax:* ectool getReleaseInventory <projectName> <releaseName>

*Example*

ectool getReleaseInventory "Default" "Production"
**getReleases**

Gets all releases.

You must specify the `projectName` argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>(Optional) A list of zero or more filter criteria definitions used to define objects to find.</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Two types of filters: &quot;property filters&quot; are used to select objects based on the value of the object's intrinsic or custom property. &quot;boolean filters&quot; (&quot;and&quot;, &quot;or&quot;, &quot;not&quot;) are used to combine one or more filters using boolean logic. Each &quot;property filter&quot; consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by ElectricFlow or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property. Property filter operators are:</td>
<td></td>
</tr>
<tr>
<td>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)</td>
<td></td>
</tr>
<tr>
<td>A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter. Boolean operators are:</td>
<td></td>
</tr>
<tr>
<td>not (1 operand) and (2 or more operands)</td>
<td></td>
</tr>
</tbody>
</table>
### Arguments

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

### Positional arguments

**projectName**

### 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*

```perl
$ec->getReleases("Default", {firstResult => 2, maxResults => 200});
```

**ectool**

*Syntax:* `ectool getReleases <projectName> [optionals...]`

*Example*

```bash
ectool getReleases "Default" --firstResult 2 --maxResults 200
```

### getStage

Retrieves a stage by name.

You must specify the `projectName` and `stageName` arguments.
### getStage

Retrieves the specified stage element.

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td>stageName</td>
<td>Name of the stage.</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline to which the stage belongs.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, stageName

**Response**

Returns the specified 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"
```

### getStages

Retrieves all the stages for one or more pipelines.

You must specify the **projectName** argument.

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline to which the stage belongs.</td>
</tr>
</tbody>
</table>
Positional arguments

    projectName

Response

Returns all the stages for one or more pipelines.

eec-perl

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

    Example

    $ec->getStages("Default", {pipelineName => "Final");

ectool

    syntax: ectool getStages <projectName> [optionals...]

    Example

    ectool getStages "Default" --pipelineName "Final"

getTask

Retrieves a task by name.

You must specify the projectName and taskName arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>taskName</td>
<td>Name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) Type of the gate.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: GateType</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline to which the stage belongs.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) Name of the stage to which this task belongs.</td>
</tr>
</tbody>
</table>

Positional arguments

    projectName, taskName

Response

Returns the specified stage element.
**getTasks**

Retrieves a task by name.

You must specify the `projectName` argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) Type of the gate.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: GateType</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline to which the stage belongs.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) Name of the stage to which this task belongs.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, taskName

**Response**

Returns the specified stage element.

**ec-perl**

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

*Example*

$ec->getStage("Default", "Check out files", {pipelineName => "Final"});

**ectool**

*Syntax:* ectool getStage <projectName> <taskName> [optionals...]

*Example*

ectool getStage "Default" "Check out files" --pipelineName "Final"
modifyDeployer

Modifies an existing deployer.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>deployerName</td>
<td>Name of the Deployer that must be unique within the project.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>newName</td>
<td>(Optional) New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Positional arguments

`projectName, deployerName`

Response

None or a status OK message.

**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"

modifyDeployerApplication

Modifies the Deployer application associated with a release.

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application. Argument Type: String</td>
</tr>
<tr>
<td>applicationProjectName</td>
<td>(Optional) The name of the project containing specified application. If not specified, it is defaulted to the release project name. Argument Type: String</td>
</tr>
<tr>
<td>orderIndex</td>
<td>(Optional) The the application deployment order, starting at 1. Argument type: Integer</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) Name of the process. Argument type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) Name of the release. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) Name of the snapshot. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, applicationName`

**Response**

Gets a Deployer application and its details.

**ec-perl**

**syntax:**

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

**Example**

```
$ec->modifyDeployerApplication("Default", "Verify versions", {applicationProjectName => "myApplications"});
```

**ectool**

**syntax:**

```
ectool modifyDeployerApplication <projectName> <applicationName> [optionals...]
```

**Example**

```
ectool modifyDeployerApplication "Default" "Verify versions" --applicationProjectName "myApplications"
```
modifyDeployerConfiguration

Modifies a Deployer configuration associated with a deployer application.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application. Argument Type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>Name of the stage of a pipeline attached to a release if specified. Argument Type: String</td>
</tr>
<tr>
<td>actualParameters</td>
<td>(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called process. Argument Type: Map</td>
</tr>
<tr>
<td>applicationProjectName</td>
<td>(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</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>(Optional) &lt;Boolean flag&gt; - 0</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment. Argument Type: String</td>
</tr>
<tr>
<td>environmentProjectName</td>
<td>(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</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) Name of the environment template.</td>
</tr>
</tbody>
</table>
modifyDeployerConfiguration

Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release. Argument Type: String</td>
</tr>
<tr>
<td>skipDeploy</td>
<td>(Optional) &lt;Boolean flag&gt; 0</td>
</tr>
</tbody>
</table>

Positional arguments

- projectName, applicationName, stageName

Response

Returns the Deployer configuration and its details.

ec-perl

 syntax:$<object>-&lt;modifyDeployerConfiguration{<projectName>, <applicationName>, <stageName>, {<optionals>}};

Example

$ec-&lt;modifyDeployerConfiguration("Default", "Shopping Cart", "Publish components", {applicationProjectName =&gt; "myApplications"});

ectool

 syntax:ectool modifyDeployerConfiguration <projectName> <applicationName> <stageName> [optionals...]

Example

ectool modifyDeployerConfiguration "Default" "Shopping Cart" "Publish components" -applicationProjectName "myApplications"

modifyNote

Modifies a note associated with an entity.

You must specify the noteName argument.

Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>noteName</td>
<td>Name of the note. Argument Type: String</td>
</tr>
</tbody>
</table>
###electricFlow

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>newName</td>
<td>(Optional) New name for the note. Argument Type: String</td>
</tr>
<tr>
<td>note</td>
<td>(Optional) Notes about the entity. Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) Project name of the entity that owns the note. Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release. Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- noteName

**Response**

Returns the selected note.

**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"
```

**modifyPipeline**

Modifies an existing pipeline.

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

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipelineName</td>
<td>Name of the pipeline that must be unique among all projects.</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>enabled</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>newName</td>
<td>(Optional) New name for an existing pipeline.</td>
</tr>
<tr>
<td>type</td>
<td>(Optional) Type of pipeline.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `projectName`, `pipelineName`

**Response**

None or a status OK message.

**ec-perl**

**syntax:**

```bash
$<object>->modifyPipeline(<projectName>, <pipelineName>, {<optionals>});
```

**Example**

```bash
$ec->modifyPipeline("Default", "Web Server Image", {newName => "Web Server");
```

**ectool**

**syntax:**

```bash
ectool modifyPipeline <projectName> <pipelineName> [optionals...]
```

**Example**

```bash
ectool modifyPipeline "Default" "Web Server Image" --newName "Web Server"
```

Back to Top
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>Name of the release. Argument Type: String</td>
</tr>
<tr>
<td>actualParameters</td>
<td>(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called process. Argument Type: Map</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code>. This text is not interpreted by the automation platform. Argument type: String</td>
</tr>
<tr>
<td>newName</td>
<td>(Optional) New name for an existing pipeline. Argument Type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline. Argument Type: String</td>
</tr>
<tr>
<td>pipelineProjectName</td>
<td>(Optional) Name of the project containing specified pipeline. If this argument is not specified, the default is the release project name. Argument Type: String</td>
</tr>
<tr>
<td>plannedEndDate</td>
<td>(Optional) The date when this release is expected to end (for example, 2006-05-15). Argument Type: String</td>
</tr>
<tr>
<td>plannedStartDate</td>
<td>(Optional) The date when this release is expected to begin (for example, 2006-05-15) Argument Type: String</td>
</tr>
</tbody>
</table>
Positional arguments

projectName, releaseName

Response

None or a status OK message.

cp-perl

syntax:

$<object>->modifyRelease(<projectName>, <releaseName>, {optionals});

Example

$ec->modifyRelease("Default", "Production", {optionals});

ectool

syntax:

ectool modifyRelease <projectName> <releaseName> [optionals]

Example

ectool modifyRelease "Default" "Production"

modifyStage

Modifies an existing stage.

You must specify the projectName and the stageName arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>Name of the stage.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>afterStage</td>
<td>(Optional) The stage that is placed after the new stage.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>beforeStage</td>
<td>(Optional) The stage that is placed before the new stage.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>
modifyStage

Modifies an existing stage.
You must specify the projectName and taskName arguments.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td>taskName</td>
<td>Name of the task.</td>
</tr>
</tbody>
</table>

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>newName</td>
<td>(Optional) New name for an existing stage.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline to which the stage belongs.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, stageName

**Response**

None or a status OK message.

**ec-perl**

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

**Example**

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

**ectool**

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

**Example**

ectool modifyStage "Default" "Preflight" --pipelineName "Daily build"

modifyTask

Modifies an existing task.
You must specify the projectName and taskName arguments.
### Arguments Descriptions

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>actualParameters</code></td>
<td>(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called process. Argument Type: Map</td>
</tr>
<tr>
<td><code>afterTask</code></td>
<td>(Optional) The task that is placed after the new task. Argument Type: String</td>
</tr>
<tr>
<td><code>approvers</code></td>
<td>(Optional) A list of task approvers who receive the notification. Argument Type: Collection</td>
</tr>
<tr>
<td><code>beforeTask</code></td>
<td>(Optional) The task that is placed before the new task. Argument Type: String</td>
</tr>
<tr>
<td><code>clearActualParameters</code></td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><code>credentials</code></td>
<td>(Optional) The credentials to be used in the task. Argument type: Collection</td>
</tr>
<tr>
<td><code>description</code></td>
<td>(Optional) Comment text describing this object; not interpreted at all by ElectricFlow. Argument Type: String</td>
</tr>
<tr>
<td><code>enabled</code></td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><code>environmentName</code></td>
<td>(Optional) Name of the environment to create from a template. Argument Type: String</td>
</tr>
<tr>
<td><code>environmentTemplateName</code></td>
<td>(Optional) Name of the environment template to use. Argument Type: String</td>
</tr>
<tr>
<td><code>errorHandling</code></td>
<td>(Optional) Type of error handling for this task. Argument Type: ErrorHandling</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>gateType</td>
<td>(Optional) The type of the gate. Argument Type: GateType</td>
</tr>
<tr>
<td>instruction</td>
<td>(Optional) Instruction associated with the task. Argument type: String</td>
</tr>
<tr>
<td>keepOnError</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>environment if an error occurs. The default is false or 0. Argument type:</td>
</tr>
<tr>
<td></td>
<td>Boolean</td>
</tr>
<tr>
<td>newName</td>
<td>(Optional) New name for an existing task. Argument Type: String</td>
</tr>
<tr>
<td>notificationTemplate</td>
<td>(Optional) String containing email formatting instructions for generating</td>
</tr>
<tr>
<td></td>
<td>notifications. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) Name of the pipeline to which the stage belongs. Argument type:</td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td>skippable</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>the pipeline. Argument type: Boolean</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) Name of the snapshot associated with the application. Argument</td>
</tr>
<tr>
<td></td>
<td>type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) Name of the stage to which the task belongs. Argument Type: String</td>
</tr>
<tr>
<td>startTime</td>
<td>(Optional) The time to begin invoking this task. The time is formatted <code>hh:mm</code></td>
</tr>
<tr>
<td></td>
<td>using the 24-hours clock (for example, 17:00). Argument Type: String</td>
</tr>
<tr>
<td>subapplication</td>
<td>(Optional) Name of the application that owns the subprocess. Argument Type:</td>
</tr>
<tr>
<td></td>
<td>String</td>
</tr>
<tr>
<td>subpluginKey</td>
<td>(Optional) If this argument references a procedure, the name of the pluginKey</td>
</tr>
<tr>
<td></td>
<td>for the procedure</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>subprocedure</code></td>
<td>(Optional) Name of the subprocedure when a procedure is referenced.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>subprocess</code></td>
<td>(Optional) Name of the ElectricFlow process.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>subproject</code></td>
<td>(Optional) Name of the project in which the procedure runs.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>subworkflowDefinition</code></td>
<td>(Optional) Name of the workflow definition in which the workflow definition is referenced.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>subworkflowStartingState</code></td>
<td>(Optional) Name of the starting state in the specified workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>taskProcessType</code></td>
<td>(Optional) The type of the process that the task can invoke.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: TaskProcessType</td>
</tr>
<tr>
<td><code>taskType</code></td>
<td>(Optional) The type of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: TaskType</td>
</tr>
<tr>
<td><code>tierResourceCounts</code></td>
<td>(Optional) The resource count per resource template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Map</td>
</tr>
<tr>
<td><code>workspaceName</code></td>
<td>(Optional) The name of the workspace.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

- `projectName, taskName`

### Response

None or a status OK message.

### `ec-perl`

#### Syntax

```perl
$s<object>->modifyTask(<projectName>, <taskName>, [<optionals>]);
```

#### Example

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

*Syntax:* `ectool modifyTask <projectName> <taskName> [optionals...]`

*Example*

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

**removeDeployerApplication**

Removes a Deployer application associated with a release.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>Name of the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>applicationProjectName</code></td>
<td>(Optional) The name of the project containing specified application. If not</td>
</tr>
<tr>
<td></td>
<td>specified, it is defaulted to the release project name.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td><code>releaseName</code></td>
<td>(Optional) Name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `applicationName`

**Response**

Gets a Deployer application and its details.

**ec-perl**

*Syntax:* `$_<object>->removeDeployerApplication{<projectName>, <applicationName>, {<optionals>}};`

*Example*

```
$ec->removeDeployerApplication("Default", "Verify versions", {applicationProjectName => "myApplications"});
```

**ectool**

*Syntax:* `ectool removeDeployerApplication <projectName> <applicationName> [optionals...]`
**removeDeployerConfiguration**

Removes a Deployer configuration associated with a deployer application.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>Name of the application.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>Name of the stage of a pipeline attached to a release if specified.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>applicationProjectName</td>
<td>(Optional) The name of the project containing the application. If this</td>
</tr>
<tr>
<td></td>
<td>argument is not specified, the default is the release project name.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- projectName
- applicationName
- stageName

**Response**

Returns the Deployer configuration and its details.

**ec-perl**

`syntax:`

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

**Example**

```perl
$ec->removeDeployerConfiguration("Default", "Shopping Cart", "Publish components", {applicationProjectName => "myApplications"});
```
**ectool**

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

**Example**

```sh
ectool removeDeployerConfiguration "Default" "Shopping Cart" "Publish components" --applicationProjectName "myApplications"
```

**runPipeline**

Runs the specified pipeline.

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

**Note:** Credential parameters are not supported in pipeline runs.

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

**Positional arguments**

`projectName, pipelineName`

**Response**

None or a status OK message.
ElectricFlow Perl API Commands

**ec-perl**

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

*Example*

$ec->runPipeline("Default", "Web Server Image", {startingState => "Green"});

**ectool**

*Syntax:* ectool runPipeline <projectName> <pipelineName> [optionals...]

*Example*

ectool runPipeline "Default" "Web Server Image" --startingState "Green"

Back to Top

**startRelease**

Starts a release.

You must specify the projectName and the releaseName arguments.

*Note:* Credential parameters are not supported in release runs.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>Name of the release. Argument Type: String</td>
</tr>
<tr>
<td>priority</td>
<td>(Optional) The priority of jobs launched by the release.. Argument Type: JobPriority</td>
</tr>
<tr>
<td>startingStage</td>
<td>(Optional) The name of the starting stage. Argument type: String</td>
</tr>
</tbody>
</table>

**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**

```sh
ectool startRelease "Default" "Production" --startingState "Checkout"
```

---

**validateDeployer**

Validate the Deployer configuration.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projectName</strong></td>
<td>Name of the project that must be unique among all projects. Argument Type: String</td>
</tr>
<tr>
<td><strong>applicationName</strong></td>
<td>Name of the application. Argument Type: String</td>
</tr>
<tr>
<td><strong>stageName</strong></td>
<td>Name of the stage of a pipeline attached to a release if specified. Argument Type: String</td>
</tr>
<tr>
<td><strong>actualParameters</strong></td>
<td>(Optional) Specifies the list of values to pass as parameters to the flow. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called process. Argument Type: Map</td>
</tr>
<tr>
<td><strong>applicationProjectName</strong></td>
<td>(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</td>
</tr>
<tr>
<td><strong>environmentName</strong></td>
<td>(Optional) Name of the environment to create from a template. Argument Type: String</td>
</tr>
<tr>
<td><strong>environmentProjectName</strong></td>
<td>(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</td>
</tr>
<tr>
<td><strong>environmentTemplateName</strong></td>
<td>(Optional) Name of the environment template to use. Argument Type: String</td>
</tr>
</tbody>
</table>
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>releaseName</td>
<td>(Optional) Name of the release. Argument Type: String</td>
</tr>
<tr>
<td>validationType</td>
<td>(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</td>
</tr>
</tbody>
</table>

### Positional arguments

projectName, applicationName, stageName

### Response

None or a status OK message.

### ec-perl

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

**Example**

```perl
$ec->validateDeployer("Default", "Make WAR file", "Checkin", {releaseName => "Web Site");
```

### ectool

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

**Example**

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

### API Commands - Plugin Management

- [createPlugin](#) on page 444
- [deletePlugin](#) on page 445
- [getPlugin](#) on page 445
- [getPlugins](#) on page 446
- [installPlugin](#) on page 447
- [modifyPlugin](#) on page 448
- [promotePlugin](#) on page 449
- [uninstallPlugin](#) on page 450
**createPlugin**

Creates a plugin from an existing project.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>Version independent name for the plugin.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>version</td>
<td>Version of the plugin.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>Name of the project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>author</td>
<td>(Optional) Name of the plugin author.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object.</td>
</tr>
<tr>
<td></td>
<td>If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags.</td>
</tr>
<tr>
<td></td>
<td>The only HTML tags allowed in the text are: ( a ) ( b ) ( br ) ( div ) ( dl ) ( font ) ( i ) ( li ) ( ol ) ( p ) ( pre ) ( span ) ( style ) ( table ) ( tc ) ( td ) ( th ) ( tr ) ( ul ) tags.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>label</td>
<td>(Optional) Label that appears in plugin lists.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginName</td>
<td>The name of the plugin you want to delete.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

Positional arguments

<table>
<thead>
<tr>
<th>pluginName</th>
</tr>
</thead>
</table>

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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginName</td>
<td>The name of the plugin to find. If the name is specified without a version number, the currently promoted version is returned if possible.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
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

getPlugins

Retrieves all installed plugins.

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

Positional arguments

None

Response

Zero or more 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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>The location of the plugin JAR file to install. If the location refers to a file on the client machine, the file will be uploaded to the server. If the location refers to a remote accessible file (for example, via an <a href="http://url">http://url</a>), the server will download it. If the location is a file: reference, the file will be read directly from the specified location on the server file system. Argument type: String</td>
</tr>
<tr>
<td>disableProjectTracking</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td>force</td>
<td>(Option) &lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

Positional arguments

url

Response

One plugin element.

ec-perl

syntax: $cmdr->installPlugin(<url>, {...});

Example

$cmdr->installPlugin("./myPlugin.jar")
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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginName</td>
<td>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</td>
</tr>
<tr>
<td>author</td>
<td>The author of the plugin. Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt; Argument type: String</td>
</tr>
<tr>
<td>label</td>
<td>The name of the plugin as displayed on the Plugins web page. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

pluginName

**Response**

One plugin element.

**ec-perl**

**syntax:** $cmdr->modifyPlugin({pluginName}, {...});
**Example**

```perl
$cmdr->modifyPlugin('TheWidget', {description => "new description"});
```

**ectool**

**syntax:** ectool modifyPlugin <pluginName> ...

**Example**

ectool modifyPlugin TheWidget --description "new description"

---

**promotePlugin**

Sets the promoted flag on a plugin. Only one version of a plugin can be promoted at a time, so setting the promoted flag to "true" on one version sets the flag to false on all other plugins with the same key. The promoted version is the one resolved by an indirect reference of the form `$/plugins/<key>` or a plugin name argument without a specified version.

You must specify the `pluginName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginName</td>
<td>The name of the plugin to promote. If the name is specified without a version number, the currently promoted version is used if possible.</td>
</tr>
<tr>
<td>promoted</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

`pluginName`

**Response**

One plugin element, which includes the plugin ID, name, time created, label, owner, key, version, project name, and so on.

**ec-perl**

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

**Example**

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

*Syntax:* `ectool promotePlugin <pluginName> ...`

**Example**

`ectool promotePlugin TheWidget-1.0`

**Back to Top**

---

**uninstallPlugin**

Uninstalls a plugin, deleting the associated project and any installed files.

You must specify the `pluginName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pluginName</code></td>
<td>The name of the plugin to uninstall. If the name is specified without a version number, the currently promoted version is used if possible. Argument type: String</td>
</tr>
<tr>
<td><code>timeout</code></td>
<td>The maximum amount of time to spend waiting for this operation to complete. Argument type: Long</td>
</tr>
</tbody>
</table>

**Positional arguments**

`pluginName`

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `cmdr-uninstallPlugin(<pluginName>, [{optionals}]);`

**Example**

`$cmdr-uninstallPlugin("TheWidget-1.0");`

**ectool**

*Syntax:* `ectool uninstallPlugin <pluginName> ...`

**Example**

`ectool uninstallPlugin TheWidget-1.0`

**Back to Top**
API Commands - Procedure Management

createProcedure on page 451
createStep on page 453
deleteProcedure on page 459
deleteStep on page 460
getProcedure on page 461
getProcedures on page 462
getStep on page 462
getSteps on page 463
modifyProcedure on page 464
modifyStep on page 466
moveStep on page 471

createProcedure

Creates a new procedure for an existing project.

You must specify projectName and procedureName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that contains this procedure. The name must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that must be unique within the project. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential specified in one of these formats:</td>
</tr>
<tr>
<td></td>
<td>• relative(for example, &quot;cred1&quot;)–The credential is assumed to be in the project that contains the requested target object.</td>
</tr>
<tr>
<td></td>
<td>• absolute(for example, &quot;/projects/BuildProject/credentials/cred1&quot;)–The credential can be from any specified project, regardless of the project for the target object.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code> Argument type: String</td>
</tr>
<tr>
<td>jobNameTemplate</td>
<td>Template used to determine the default name of jobs launched from a procedure. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the default resource or pool used in steps run by this procedure. Argument type: String</td>
</tr>
<tr>
<td>timeLimit</td>
<td>If no time limit was specified on the calling step, time limits are copied to the calling step from the procedure. If the procedure is called from runProcedure (or a schedule), the time limit acts as a global job timeout. The timer for the procedure starts as soon as the calling step or job becomes runnable (all preconditions are satisfied). Argument type: String</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>Time limit units are hours</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the default workspace used in steps run by this procedure. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `procedureName`

**Response**

None or status OK message.

**ec-perl**

*Syntax:* $cmdr-&gt;createProcedure(<projectName>, <procedureName>, {<optionals>});

*Example*

$cmdr-&gt;createProcedure("Test Proj", "Run Build", {resourceName =&gt; "Test Resource"});

**ectool**

*Syntax:* ectool createProcedure <projectName> <procedureName> ...
createStep

**Example**

ectool createProcedure "Test Proj" "Run Build" --resourceName "Test Resource"

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of a project that must be unique among all projects.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name for a procedure that must be unique within the project.</td>
</tr>
<tr>
<td>stepName</td>
<td>Name of the step that must be unique within the procedure.</td>
</tr>
<tr>
<td>actualParameters</td>
<td>(Optional) Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure. For more information about parameters, click here.</td>
</tr>
<tr>
<td>alwaysRun</td>
<td>(Optional) The value for alwaysRun is a &lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

Argument type: String

Argument type: String

Argument type: String

Argument type: Map

Argument type: Boolean
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>broadcast</td>
<td>(Optional) The broadcast value is `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>command</td>
<td>(Optional) The command to run. This argument is applicable only to command steps. Argument type: String</td>
</tr>
<tr>
<td>commandFile</td>
<td>(Optional) This option is supported only in Perl and ectool bindings - it is not a part of the XML protocol. Contents of the command file is read and stored in the &quot;command&quot; field. This is an alternative argument for command and is useful if the &quot;command&quot; field spans multiple lines. The commandFile value is the actual command file text. This argument is applicable to command steps only.</td>
</tr>
<tr>
<td>condition</td>
<td>(Optional) If empty or non-zero, the step will run. If set to &quot;0&quot;, the step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps. Argument type: String</td>
</tr>
</tbody>
</table>
| credentialName | (Optional) The credential to use for impersonation on the agent. Name of the credential in one of these forms:  
|               | - relative (for example, "cred1")– The credential is assumed to be in the project that contains the request target object.  
|               | - absolute (for example, "/projects/BuildProject/credentials/cred1")– The credential can be from any specified project, regardless of the target object’s project. Argument type: String |
| description | (Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> <b> <br> <div> <dl> <font> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul> Argument type: String |
### Arguments | Descriptions
---|---

<table>
<thead>
<tr>
<th>errorHandling</th>
<th>(Optional) Determines what happens to the procedure if the step fails:</th>
</tr>
</thead>
<tbody>
<tr>
<td>failProcedure</td>
<td>The current procedure continues, but the overall status is error (default).</td>
</tr>
<tr>
<td>abortProcedure</td>
<td>Aborts the current procedure, but allows already-running steps in the current procedure to complete.</td>
</tr>
<tr>
<td>abortProcedureNow</td>
<td>Aborts the current procedure and terminates running steps in the current procedure.</td>
</tr>
<tr>
<td>abortJob</td>
<td>Aborts the entire job, terminates running steps, but allows alwaysRun steps to run.</td>
</tr>
<tr>
<td>abortJobNow</td>
<td>Aborts the entire job and terminates all running steps, including alwaysRun steps.</td>
</tr>
<tr>
<td>ignore</td>
<td>Continues as if the step succeeded.</td>
</tr>
</tbody>
</table>

Argument type: ErrorHandling

| exclusive | (Optional) The value for exclusive is a `<Boolean flag - 0|1|true|false>`. |
|---|---|
| If set to 1, this step should acquire and retain this resource exclusively. The default is false. |

Note: When you set exclusive, exclusiveMode is set to "job".

Argument type: Boolean

<table>
<thead>
<tr>
<th>exclusiveMode</th>
<th>(Optional) Use one of the following options:</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>the &quot;default&quot;, which does not retain a resource.</td>
</tr>
<tr>
<td>Job</td>
<td>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 &quot;Release Exclusive&quot; 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.</td>
</tr>
<tr>
<td>Step</td>
<td>keeps the resource for the duration of the step.</td>
</tr>
<tr>
<td>Call</td>
<td>keeps the resource for the duration of the procedure that called this step, which is equivalent to 'job' for top level steps.</td>
</tr>
</tbody>
</table>

Argument type: ExclusiveMode

<table>
<thead>
<tr>
<th>logFileName</th>
<th>(Optional) A custom log file name produced by running the step. By default, ElectricFlow assigns a unique name for this file.</th>
</tr>
</thead>
</table>

Argument type: String
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel</td>
<td>(Optional) The value for <code>parallel</code> is a `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>postProcessor</td>
<td>(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 &quot;postp&quot; 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 (cmd for Windows, sh for UNIX). Argument type: String</td>
</tr>
<tr>
<td>precondition</td>
<td>(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 <code>precondition</code> is evaluated. A <code>precondition</code> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE. Precondition example: Assume we defined these 4 steps: 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system Step 1 is an ordinary serial step. Steps 2 and 3 can run in parallel because they depend only on step 1's completion. Step 4 depends on step 2, but not step 3. You can achieve optimal step execution order with preconditions: • Make steps 2-4 run in parallel. • Step 2 needs a job property set at the end of its step to indicate step 2 is completing (<code>/myJob/buildInstallerCompleted=1</code>). • Set a precondition in step 4: <code>$[/myJob/buildInstallerCompleted]</code> Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| releaseExclusive  | (Optional) <Boolean flag> - 0|1|true|false>  
|                   | Declares whether or not this step will release its resource, which is currently held exclusively.  
|                   | **Note:** Setting this flag to "true" is the same as setting releaseMode to release.  
|                   | Argument type: Boolean                                                     |
| releaseMode       | (Optional) Use one of the following options:  
|                   | - none - the "default" - no action if the resource was not previously marked as "retain."  
|                   | - release - releases the resource at the end of this step. If the resource for the step was previously acquired with "Retain exclusive" (either by this step or some preceding step), the resource exclusivity is canceled at the end of this step. The resource is released in the normal way so it may be acquired by other jobs.  
|                   | - releaseToJob - allows a step to promote a "step exclusive" resource to a Job exclusive resource.  
|                   | Argument type: ReleaseMode                                                 |
| resourceName      | (Optional) Name for the resource that must be unique among all resources.  
|                   | Argument type: String                                                      |
| shell             | (Optional) Where shell is the name of a program used to execute commands contained in the "command" field. The name of a temporary file containing commands will be appended to the end of this invocation line. Normally, this file is a command shell, but it can be any other command line program. The default is to use the standard shell for the platform it runs on (cmd for Windows, sh for UNIX). This is applicable to command steps only.  
|                   | Argument type: String                                                      |
| subprocedure      | (Optional) The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the command or commandFile options.  
|                   | Argument type: String                                                      |
| subproject        | (Optional) If a subprocedure argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.  
|                   | Argument type: String                                                      |
Arguments | Descriptions
---|---
timeLimit | (Optional) The maximum length of time the step is allowed to run. After the time specified, the step will be aborted. The time limit is specified in units that can be hours, minutes, or seconds. Argument type: String
timeLimitUnits | (Optional) Specify hours|minutes|seconds for time limit units. Argument type: TimeLimitUnits
workingDirectory | (Optional) The ElectricFlow agent sets this directory as the “current working directory,” when running the command contained in the step. If no working directory is specified in the step, ElectricFlow uses the directory it created for the job in the ElectricFlow workspace as the working directory. 

**Note:** If running a step on a proxy resource, this directory must exist on the proxy target. Argument type: String
workspaceName | (Optional) The name of the workspace where the log files for this step will be stored. Argument type: String

**Positional arguments**

- `projectName`, `procedureName`, `stepName`

**Response**

None or a status OK message.

**ec-perl**

**syntax:** $cmdr->createStep(<projectName>, <procedureName>, <stepName>, {
	<optionals>});

Specifying most arguments to the Perl `createStep` API is fairly intuitive. Similar to any other API, key-value pairs are specified in a hash argument for all optional parameters. However, specifying actual parameters is a little different because they are not arbitrary key-values characterizing the step. Actual parameters are key-values characterizing actual parameters to the step. See the following `createStep` request in XML:

```xml
<createStep>
	<projectName>MyProject</projectName>
	<procedureName>MyProcedure</procedureName>
	<stepName>Step1</stepName>
	<actualParameter>
		<actualParameterName>parm1</actualParameterName>
		<value>myval</value>
	</actualParameter>
	<actualParameter>
		<actualParameterName>parm2</actualParameterName>
	</actualParameter>
</createStep>
```
Each actual parameter key-value is under an `<actualParameter>` element, which is codified in the optional arguments hash in the Perl API like this:

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

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

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

**Example**

```perl
$cmdr->createStep("Test Proj", "Run Build", "Common Cleanup", {subprocedure => "Delay",
  actualParameter => {actualParameterName => 'Delay Time', value => '5'});}
```

**ectool**

**syntax:** `ectool createStep <projectName> <procedureName> <stepName> ...`

Specifying actual parameters in an `ectool` call is also different than specifying other arguments. Specify each key-value as an equal-sign delimited value:

```perl
ectool createStep ... --actualParameter "Delay Time=5" "parm2=val2"
```

**Note:** If the parameter name or value contains spaces, quotes are needed.

**Examples**

```perl
ectool createStep "Test Proj" "Run Build" "Compile" --command "make all"

ectool createStep "Test Proj" "Run Build" "Common Cleanup" --subprocedure "Delay"
  --actualParameter "Delay Time=5"
```

**deleteProcedure**

Deletes a procedure, including all steps.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project that contains this procedure. The name must be unique within the project. Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>The name of the procedure that must be unique within the project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

- projectName, procedureName

### Response

None or a status OK message.

#### ec-perl

**syntax:** `$cmdr->deleteProcedure(<projectName>, <procedureName>);`

**Example**

```
$cmdr->deleteProcedure("Test Proj", "Run Build");
```

#### ectool

**syntax:** `ectool deleteProcedure <projectName> <procedureName>`

**Example**

```
ectool deleteProcedure "Test Proj" "Run Build"
```
Response
None or a status OK message.

ec-perl

**syntax:** $cmdr->deleteStep(<projectName>, <procedureName>, <stepName>);

**Example**
$cmdr->deleteStep("Test Proj", "Run Build", "Compile");

ectool

**syntax:** ectool deleteStep <projectName> <procedureName> <stepName>

**Example**
ectool deleteStep "Test Proj" "Run Build" "Compile"

getProcedure
Finds a procedure by its name.
You must specify a **projectName** and a **procedureName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that contains this procedure. The name must be unique within the project. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that must be unique within the project. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**
projectName, procedureName

Response
One procedure element, which includes the procedure ID, name, time created, job name template, owner, resource name, workspace name, project name, and so on.

ec-perl

**syntax:** $cmdr->getProcedure(<projectName>, <procedureName>);

**Example**
$cmdr->getProcedure("Test Proj", "Run Build");

ectool

**syntax:** ectool getProcedure <projectName> <procedureName>
**getProcedures**

Retrieves all procedures in one project.

You must specify the `projectName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that contains this procedure. The name must be unique within the project. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- **projectName**

**Response**

One or more `procedure` elements.

**ec-perl**

**syntax:** `cmdr->getProcedures(<projectName>);`

**Example**

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

**ectool**

**syntax:** `ectool getProcedures <projectName>`

**Example**

```
ectool getProcedures "Test Proj"
```

**getStep**

Retrieves a step from a procedure.

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

---

**Example**

```
ectool getProcedure "Test Proj" "Run Build"
```
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that contains this procedure. The name must be unique within the project. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that must be unique within the project. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step that must be unique within the procedure. Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

- projectName, procedureName, stepName

### Response

One step element.

#### ec-perl

**syntax:**

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

**Example**

```
$cmdr->getStep("Test Proj", "Run Build", "Compile");
```

#### ectool

**syntax:**

```
ectool getStep <projectName> <procedureName> <stepName>
```

**Example**

```
ectool getStep "Test Proj" "Run Build" "Compile"
```

[Back to Top](#)

## getSteps

Retrieves all steps in a procedure.

You must specify the projectName and procedureName.

### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that contains this procedure. The name must be unique within the project. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that must be unique within the project. Argument type: String</td>
</tr>
</tbody>
</table>
**Positional arguments**

    projectName, procedureName

**Response**

Zero or more `step` elements.

**ec-perl**

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

*Example*

$cmdr->getSteps("Test Proj", "Run Build");

**ectool**

*syntax:* ectool getSteps <projectName> <procedureName>

*Example*

ectool getSteps "Test Proj" "Run Build"

**modifyProcedure**

Modifies an existing procedure.

You must specify `projectName` and `procedureName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that contains this procedure. The name must be unique within the project. You must also enter <code>procedureName</code>. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that must be unique within the project. You must also enter <code>projectName</code>. Argument type: String</td>
</tr>
</tbody>
</table>
| credentialName | Name of the credential in one of these forms:  
  - relative (for example, "cred1")–The credential is assumed to be in the project that contains the request target object.  
  - absolute (for example, 
    "projects/BuildProject/credentials/cred1")–The credential can be from any specified project, regardless of the target object’s project.  
  Argument type: String |
## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>description</strong></td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: `&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;dt&gt; &lt;dd&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tbody&gt; &lt;tr&gt; &lt;td&gt; &lt;th&gt; &lt;ul&gt; Argument type: String</td>
</tr>
<tr>
<td><strong>jobNameTemplate</strong></td>
<td>Job name format for jobs created by running this procedure. Argument type: String</td>
</tr>
<tr>
<td><strong>newName</strong></td>
<td>New name of the procedure. Argument type: String</td>
</tr>
<tr>
<td><strong>resourceName</strong></td>
<td>The name of the default resource where steps belonging to this procedure will run. This name can be a resource pool name. Argument type: String</td>
</tr>
<tr>
<td><strong>timeLimit</strong></td>
<td>If no time limit was specified on the calling step, time limits are copied to the calling step from the procedure. If the procedure is called from <code>runProcedure</code> (or a schedule), the time limit acts as a global job timeout. The &quot;timer&quot; for the procedure starts as soon as the calling step/job becomes runnable (all preconditions are satisfied). Argument type: String</td>
</tr>
<tr>
<td><strong>timeLimitUnits</strong></td>
<td>Time limit units are hours</td>
</tr>
<tr>
<td><strong>workspaceName</strong></td>
<td>The name of the default workspace where job output is stored. Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

`projectName, procedureName`

### Response

None or a status OK message.

### ec-perl

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

**Example**

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

*Syntax:* `ectool modifyProcedure <projectName> <procedureName> ...`

**Example**

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

**modifyStep**

Modifies an existing step.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project that contains this procedure. The name must be unique within the project. You must also enter <code>procedureName</code>.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>procedureName</code></td>
<td>The name of the procedure that must be unique within the project. You must also enter <code>projectName</code>.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>stepName</code></td>
<td>The name of the step that must be unique within the procedure. You must also enter <code>projectName</code> and <code>procedureName</code>.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>actualParameter</code></td>
<td>Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called procedure.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Map</td>
</tr>
<tr>
<td><code>alwaysRun</code></td>
<td>`&lt;Boolean flag = 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td><code>broadcast</code></td>
<td>`&lt;Boolean flag = 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| clearActualParameters | `<Boolean flag - 0|1|true|false>`  
If set to true or 1, all the actual parameters will be removed from the step.  
Argument type: Boolean |
| command              | The step command.  
Argument type: String |
| commandFile          | This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. The contents of the command file is read and stored in the "command" field. This is an alternative argument for command and is useful if the "command" field spans multiple lines. |
| condition            | If empty or non-zero, the step will run. If set to "0", the step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps.  
Argument type: String |
| credentialName       | Name of the credential in one of these forms:  
  - relative (for example, "cred1")–The credential is assumed to be in the project that contains the request target object.  
  - absolute (for example, "projects/BuildProject/credentials/cred1")–The credential can be from any specified project, regardless of the target object’s project.  
Argument type: String |
| description           | A plain text or HTML description for this object. If using HTML, you must surround your text with `<html> ... </html>` tags. The only HTML tags allowed in the text are: `<a> `<b> `<br> `<div> `<dl> `<font> `<i> `<li> `<ol> `<p> `<pre> `<span> `<style> `<table> `<tt> `<td> `<th> `<tr> `<ul>  
Argument type: String |
### Arguments

| errorHandling | Determines what happens to the procedure if the step fails:  
failProcedure - The current procedure continues, but the overall status is error (default).  
  - abortProcedure - Aborts the current procedure, but allows already-running steps in the current procedure to complete.  
  - abortProcedureNow - Aborts the current procedure and terminates running steps in the current procedure.  
  - abortJob - Aborts the entire job, terminates running steps, but allows alwaysRun steps to run.  
  - abortJobNow - Aborts the entire job and terminates all running steps, including alwaysRun steps.  
  - ignore - Continues as if the step succeeded.  
Argument type: ErrorHandling |
| exclusive | The value for exclusive is a `<Boolean flag -0|1|true|false>`.  
If set to 1, this indicates this step should acquire and retain this resource exclusively. The default is false.  
When you set exclusive, exclusiveMode is set to job.  
Argument type: Boolean |
| exclusiveMode | Use one of the following options:  
  - None - the "default", which does not retain a resource.  
  - Job - keeps the resource for the duration of the job. No other job can use this resource, regardless of its step limit, until this job completes or "Release Exclusive" is used in a step. Future steps for this job will use this resource in preference to other resources—if this resource meets the needs of the steps and its step limit is not exceeded.  
  - Step - keeps the resource for the duration of the step.  
  - Call - keeps the resource for the duration of the procedure that called this step, which is equivalent to 'job' for top level steps.  
Argument type: ExclusiveMode |
| logFileName | A custom log file name produced by running the step. By default, ElectricFlow assigns a unique name to this file.  
Argument type: String |
| newName | New name of the step.  
Argument type: String |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>parallel</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>precondition</td>
<td>By default, if the step has no precondition, it will run when scheduled. Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a <code>precondition</code> is evaluated. A <code>precondition</code> is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE. Precondition example: Assume we defined these 4 steps: 1. Build object files and executables 2. Build installer 3. Run unit tests 4. Install bits on test system Step 1 is an ordinary serial step. Steps 2 and 3 can run in parallel because they depend only on step 1's completion. Step 4 depends on step 2, but not step 3. You can achieve optimal step execution order with preconditions: • Make steps 2-4 run in parallel. • Step 2 needs a job property set at the end of its step to indicate step 2 is completing (/myJob/buildInstallerCompleted=1). • Set a precondition in step 4: $[/myJob/buildInstallerCompleted] Argument type: String</td>
</tr>
<tr>
<td>postProcessor</td>
<td>The name of a program to run (script) after a step completes. This program looks at the step output to find errors and warnings. ElectricFlow includes a customizable program called &quot;postp&quot; for this purpose. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| releaseExclusive| *<Boolean flag - 0|1|true|false>*  
This declares whether or not this step will release its resource, which is currently held exclusively.  
**Note:** Setting this flag to "true" is the same as setting releaseMode to "release".  
Argument type: Boolean |
| releaseMode     | Use one of the following options:  
- none - the "default" - no action if the resource was not previously marked as "retain."  
- release - releases the resource at the end of this step. If the resource for the step was previously acquired with "Retain exclusive" (either by this step or some preceding step), the resource exclusivity is canceled at the end of this step. The resource is released in the normal way so it may be acquired by other jobs.  
- releaseToJob - allows a step to promote a Step exclusive resource to a Job exclusive resource.  
Argument type: ReleaseMode |
| resourceName    | The name of the resource used by this step. The name must be unique among all resources.  
Argument type: String |
| shell           | Where shell is the name of a program used to execute commands contained in the "command" field. The name of a temporary file containing commands will be appended to the end of this invocation line.  
Normally, this file is a command shell, but it could be any other command line program. The default is to use the standard shell for the platform it runs on.  
Argument type: String |
| subprocedure    | The name of the nested procedure to call when this step runs. If a subprocedure is specified, do not include the command or commandField.  
Argument type: String |
| subproject      | If a subprocedure argument is used, this is the name of the project where that subprocedure is found. By default, the current project is used.  
Argument type: String |
Arguments | Descriptions
---|---
**timeLimit** | The maximum length of time the step is allowed to run. After the time specified, the step will be aborted. Argument type: String
**timeLimitUnits** | Units for step time limit: <hours|minutes|seconds> Argument type: TimeLimitUnits
**workingDirectory** | The ElectricFlow agent sets this directory as the "current working directory," running the command contained in the step. If no working directory is specified in the step, ElectricFlow uses the directory it created for the job in the ElectricFlow workspace. Argument type: String
**workspaceName** | 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("Test Proj", "Run Build", "Compile", {commandFile => "tempfile.txt"});
```

**ectool**

*Syntax:* `ectool modifyStep <projectName> <procedureName> <stepName> ...
```

*Example*
```
ectool modifyStep "Test Proj" "Run Build" "Compile" --commandFile tempfile.txt
```

**moveStep**
Moves a step within a procedure.

*You must specify* **projectName, procedureName, and stepName.**
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project that contains this procedure. The name must be unique within the project. You must also enter procedureName. Argument type: String</td>
</tr>
<tr>
<td><code>procedureName</code></td>
<td>The name of the procedure that must be unique within the project. You must also enter <code>projectName</code>. Argument type: String</td>
</tr>
<tr>
<td><code>stepName</code></td>
<td>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</td>
</tr>
<tr>
<td><code>beforeStep</code></td>
<td>Moves the step (stepName) to position before the step &quot;named&quot; by this option. If omitted, stepName is moved to the end of the list of steps. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `projectName`, `procedureName`, `stepName`

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* 

```
$cmdr->moveStep(<projectName>, <procedureName>, <stepName>, {<optionals>});
```

**Example**

```
$cmdr->moveStep("Test Proj", "Run Build", "Get Sources", {beforeStep => "Compile"});
```

**ectool**

*Syntax:*

```
ectool moveStep <projectName> <procedureName> <stepName> ...
```

**Example**

```
ectool moveStep "Test Proj" "Run Build" "Get Sources"
--beforeStep "Compile"
```

### API Commands - Process

- `createProcess` on page 473
- `deleteProcess` on page 474
- `getProcess` on page 475
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.
  
  **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.

defeat

**Syntax:**

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

**Example:**

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

defeatool

**Syntax:**

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

**Example:**

```bash
ectool createProcess default newProcess --componentName VCScomponent
```

defeatProcess

Deletes an application or component process.

**Required Arguments**

- **projectName**

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

  **Argument Type:** String

- **processName**

  **Description:** Name of the process.

  **Argument Type:** String

**Optional Arguments**

- **applicationName**

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

  **Argument Type:** String
componentApplicationName

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

**Argument Type:** String

componentName

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

**Argument Type:** String

Response

None or a status OK message.

dc-perl

Syntax:

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

Example:

```perl
$ec->deleteProcess("default", "newProcess", {componentName => "Component1"});
```

dc-tool

Syntax:

```bash
dc-tool deleteProcess <projectName> <processName> [optionals...]
```

Example:

```bash
dc-tool deleteProcess default newProcess --componentName Component1
```

### getProcess

Retrieves an application or component process.

**Required Arguments**

**projectName**

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

**Argument Type:** String

**processName**

**Description:** Name of the process.

**Argument Type:** String

**Optional Arguments**

**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.
**getProcesses**

Retrieves all processes in an application or component.

**Required Arguments**

**projectName**

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

*Argument Type:* String

**Optional Arguments**

**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**

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

**Argument Type:** String

**Response**

Retrieves the specified process element.

**ec-perl**

**Syntax:**

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

**Example:**

```perl
Sec->getProcess("default", "newProcess", {componentName => "VCS"});
```

**ectool**

**Syntax:**

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

**Example:**

```bash
ectool getProcess default newProcess --componentName VCScomponent
```

**applicationName**

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

**Argument Type:** String

**componentApplicationName**

**Description:** Application name of the component, if the component is scoped to application.

**Argument Type:** String

**Response**

Retrieves zero or more process elements.

**ec-perl**

**Syntax:**

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

**Example:**

```perl
Sec->getProcesses("default", {componentName => "VCS"});
```

**ectool**

**Syntax:**
ectool getProcesses <projectName> [optionals...]

Example:

ectool getProcesses default --componentName VCScomponent

modifyProcess

Modifies an existing process.

**Required Arguments**

**projectName**

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

*Argument Type:* String

**processName**

*Description:* Name of the process.

*Argument Type:* String

**Optional Arguments**

**applicationName**

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

*Argument Type:* String

**componentApplicationName**

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

*Argument Type:* String

**componentName**

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

*Argument Type:* String

**credentialName**

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

*Argument Type:* String

**description**

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

**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:**

```perl
$ec->modifyProcess("default", "newProcess", {componentName => "VCS", newName => "VCScomponent", description => "An updated description"});
```

**ectool**

**Syntax:**

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

**Example:**

```bash
ectool modifyProcess default newProcess --componentName VCScomponent --newName VCS --description "A description"
```

**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

environmentTemplateName
   Description: Name of the environment template.
   Argument Type: String

environmentTemplateTierMapName
   Description: Environment template tier map name.
   Argument Type: String

keepOnError
   Description: Set this flag to "true" to keep the environment when an error occurs.
   The keepOnError value = \<Boolean flag\> -0|1|true|false>. Defaults to false or 0.
   Argument Type: Boolean

environmentTemplateTierMapName
   Description: Name of the environment template tier.
   Argument Type: String

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

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

validate

Description: Validates that the application process, tier map, and environment are well-defined and valid before the running the application process. `<Boolean flag -0|1|true|false>`. The default is true.

Argument Type: Boolean

Response

Returns new job ID.

ec-perl

Syntax:

```perl
$obj->runProcess(<projectName>, <applicationName>, <processName>, {<optionals}>);
```

Example:

```perl
Sec->runProcess("default", "NewApp", "newProcess", {destinationProject => "deploy1");
```

ectool

Syntax:

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

Example:

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

API Commands - Process Dependency

- `createProcessDependency` on page 482
- `deleteProcessDependency` on page 483
- `getProcessDependencies` on page 484
modifyProcessDependency on page 485

createProcessDependency

Creates a dependency between two process steps.

**Required Arguments**

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

- `processName`
  - **Description**: Name of the process.
  - **Argument Type**: String

- `processStepName`
  - **Description**: Name of the process step.
  - **Argument Type**: String

- `targetProcessStepName`
  - **Description**: Name of the target process step.
  - **Argument Type**: String

**Optional Arguments**

- `applicationName`
  - **Description**: Name of the application, if the process is owned by an application; must be unique among all projects.
  - **Argument Type**: String

- `branchCondition`
  - **Description**: Condition of the branch.
  - **Argument Type**: String

- `branchConditionName`
  - **Description**: Name of the branch condition.
  - **Argument Type**: String

- `branchConditionType`
  - **Description**: Type of the branch condition.
  - **Argument Type**: BranchConditionType

- `branchType`
  - **Description**: Type of the branch.
  - **Argument Type**: BranchType
componentApplicationName

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

**Argument Type:** String

cOMPONENTNAME

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

**Argument Type:** String

**Response**

Returns a process dependency element.

e-c-perl

**Syntax:**

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

**Example:**

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

ectool

**Syntax:**

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

**Example:**

```
ectool createProcessDependency default newProcess "Step A" "Step B" --component Name VCScomponent
```

deleteProcessDependency

**Description:** Deletes a dependency between two process steps.

**Required Arguments**

**projectName**

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

**Argument Type:** String

**processName**

**Description:** Name of the process.

**Argument Type:** String

**processStepName**

**Description:** Name of the process step.

**Argument Type:** String

**targetProcessStepName**

**Description:** Name of the target process step.
Description: Name of the target process step.

Argument Type: String

Optional Arguments

applicationName

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

Argument Type: String

componentApplicationName

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

Argument Type: String

componentName

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

Argument Type: String

Response

None or a status OK message.

ec-perl

Syntax:

$<object>-deleteProcessDependency(<projectName>, <processName>, <processStepName>, <targetProcessStepName>, [<optionals>]);

Example:


ectool

Syntax:

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

Example:

ectool deleteProcessDependency default newProcess "Step B" "Step C" --componentName VCScomponent

getProcessDependencies

Retrieves all dependencies for a process.

Required Arguments

projectName

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

Argument Type: String
processName
   Description: Name of the process.
   Argument Type: String

Optional Arguments

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", "newProcess", {componentName => "VCScomponent"});

ectool
   Syntax:
     ectool getProcessDependencies <projectName> <processName> [optionals...]
   Example:
     ectool getProcessDependencies default newProcess --componentName VCScomponent

modifyProcessDependency

   Modifies a dependency between two process steps.

   Required Arguments

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

processName
Description: Name of the process.
Argument Type: String

processStepName
Description: Name of the process step.
Argument Type: String

targetProcessStepName
Description: Name of the target process step.
Argument Type: String

Optional Arguments

applicationName
Description: Name of the application, if the process is owned by an application.
Argument Type: String

branchCondition
Description: Condition of the branch.
Argument Type: String

branchConditionName
Description: Name of the branch condition.
Argument Type: String

branchConditionType
Description: Type of the branch condition.
Argument Type: BranchConditionType

branchType
Description: Type of the branch.
Argument Type: BranchType

cOMPONENTAPPLICATIONNAME
Description: If specified, the component is scoped to this application, not the project.
Argument Type: STRING

componentName
Description: Name of the component, if the process is owned by a component.
Argument Type: String
**ec-perl**

Syntax:

```perl
$<object>-modifyProcessDependency(<projectName>, <processName>, <processStepName>, <targetProcessStepName>, [<optionals>]);
```

Example:

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

**ectool**

Syntax:

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

Example:

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

### API Commands - Process Step

- **createProcessStep** on page 487
- **deleteProcessStep** on page 490
- **getProcessStep** on page 491
- **getProcessSteps** on page 492
- **modifyProcessStep** on page 493

**Note:** Several of the following API commands have context type optional arguments. For example, a step command may reference either a procedure or component.

---

**createProcessStep**

Creates a new process step.

**Required Arguments**

- **projectName**

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

  **Argument Type:** String

- **processName**

  **Description:** Name of the process.

  **Argument Type:** String

- **processStepName**

  **Description:** Name of the process step.

  **Argument Type:** String
**Optional Arguments**

- **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

- **beforeProcessStep**
  - **Description:** If specified, the process step will be placed before the named process step.
  - **Argument Type:** String

- **componentName**
  - **Description:** Name of the component, if the process is owned by a component.
  - **Argument Type:** String

- **credentialName**
  - **Description:** Name of the credential object.
  - **Argument Type:** String

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

- **processStepType**
**Description:** Defines type of the process step.
**Argument Type:** ProcessStepType

*subcomponent*

**Description:** If referencing a component process, the name of the component.
**Argument Type:** String

*subcomponentProcess*

**Description:** If referencing a component process, the name of the component process.
**Argument Type:** String

*subprocedure*

**Description:** If referencing a procedure, the name of the procedure.
**Argument Type:** String

*subproject*

**Description:** If referencing a procedure, the name of the procedure's project.
**Argument Type:** String

*timeLimit*

**Description:** Maximum amount of time that the step can execute; abort if it exceeds this time.
**Argument Type:** String

*timeLimitUnits*

**Description:** Units for the step time limit: seconds, minutes, or hours.
**Argument Type:** TimeLimitUnits

workspaceName

**Description:** Name of the workspace.
**Argument Type:** String

**Response**

Returns a process step element.

**ec-perl**

**Syntax:**

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

**Example:**

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

**ectool**

**Syntax:**

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

```plaintext
ectool createProcessStep default newProcess "Step A" --componentName VCScomponent
```

deleteProcessStep

Deletes an application or component process step.

**Required Arguments**

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

- `processName`
  
  **Description:** Name of the process.
  
  **Argument Type:** String

- `processStepName`
  
  **Description:** Name of the process step.
  
  **Argument Type:** String

**Optional Arguments**

- `applicationName`
  
  **Description:** Name of the application, if the process is owned by an application; must be unique among all projects.
  
  **Argument Type:** String

- `componentApplicationName`
  
  **Description:** If specified, the component is scoped to this application, not the project.
  
  **Argument Type:** String

- `componentName`
  
  **Description:** Name of the component, if the process is owned by a component.
  
  **Argument Type:** String

**Response**

None or a status OK message.

**ec-perl**

**Syntax:**

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

**Example:**
$ec->deleteProcessStep ("default", "newProcess", "stepToDelete", {componentName => "VCScomponent"});

ectool

Syntax:

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

Example:

ectool deleteProcessStep default newProcess "stepToDelete" --componentName VCS component

getProcessStep

Gets an application or component process step.

**Required Arguments**

**projectName**

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

*Argument Type:* String

**processName**

*Description:* The name of the process.

*Argument Type:* String

**processStepName**

*Description:* The name of the process step.

*Argument Type:* String

**Optional Arguments**

**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 the specified process step element.

ec-perl
Syntax:

\$<object>\rightarrow \text{getProcessStep}(<\text{projectName}>, <\text{processName}>, <\text{processStepName}>, [{\text{optionals}}]);

Example:

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

ectool
Syntax:

ectool getProcessStep <\text{projectName}> <\text{processName}> <\text{processStepName}> [optionals ...]

Example:

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

getProcessSteps
Retrieves all the process steps in an application or component process.

Required Arguments

\text{projectName}

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

Argument Type: String

\text{processName}

Description: Name of the process.

Argument Type: String

Optional Arguments

\text{applicationEntityRevisionId}

Description: The revision ID of the versioned project.

Argument Type: UUID

\text{applicationName}

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

Argument Type: String

\text{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.

**Response**

Retrieves zero or more process step elements.

**ec-perl**

**Syntax:**

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

**Example:**

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

**ectool**

**Syntax:**

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

**Example:**

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

**modifyProcessStep**

Modifies an existing process step.

**Required Arguments**

**projectName**

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

**Argument Type:** String

**processName**

**Description:** Name of the process.

**Argument Type:** String

**processStepName**

**Description:** Name of the process step.

**Argument Type:** String

**Optional Arguments**

**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; must be unique among all projects.

**Argument Type:** String

applicationTierName

**Description:** Name of the application tier on which to run the step.

**Argument Type:** String

beforeProcessStep

**Description:** If specified, the process step will be placed before the named process step.

**Argument Type:** String

clearActualParameters

**Description:** True if the step should remove all actual parameters.

**Argument Type:** Boolean

componentApplicationName

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

**Argument Type:** String

componentName

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

**Argument Type:** String

credentialName

**Description:** Name of the credential object.

**Argument Type:** String

description

**Description:** Comment text describing this object; not interpreted at all by 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
newName
   **Description:** New name for an existing object that is being renamed.
   **Argument Type:** String

processStepType
   **Description:** Defines type of the process step.
   **Argument Type:** ProcessStepType

subcomponent
   **Description:** If referencing a component process, the name of the component.
   **Argument Type:** String

subcomponentApplicationName
   **Description:** If referencing a component process, the name of the component application (if it has not been scoped to a project).
   **Argument Type:** String

subcomponentProcess
   **Description:** If referencing a component process, the name of the component process.
   **Argument Type:** String

subprocedure
   **Description:** If referencing a procedure, the name of the procedure.
   **Argument Type:** String

subproject
   **Description:** If referencing a procedure, the name of the procedure's project.
   **Argument Type:** String

timeLimit
   **Description:** Maximum amount of time that the step can execute; abort if it exceeds this time.
   **Argument Type:** String

timeLimitUnits
   **Description:** Units for the step time limit: seconds, minutes, or hours.
   **Argument Type:** TimeLimitUnits

workspaceName
   **Description:** Name of the workspace.
   **Argument Type:** String

**Response**
   Retrieves an updated process step element.

**ec-perl**
   **Syntax:**
modifyProcessStep

Example:

```bash
Sec->modifyProcessStep("default", "newProcess", "Step 1", {componentName => "VC Scomponent", newName => "Step 2", description => "A description"});
```

ectool

Syntax:

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

Example:

```
ectool modify ProcessStep newProcess "Step A" --componentName VCScomponent --new Name "Step B" --description "A description"
```

API Commands - Project Management

- **createProject** on page 496
- **deleteProject** on page 497
- **getProject** on page 498
- **getProjects** on page 499
- **modifyProject** on page 499
- **reloadSetupScripts** on page 501

**createProject**

Creates a new project.

You must specify a **projectName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique within the project.</td>
</tr>
<tr>
<td>credentialName</td>
<td>Name of the credential in one of these forms:</td>
</tr>
<tr>
<td></td>
<td>- <strong>relative</strong> <em>(for example, &quot;cred1&quot;)</em>—The credential is assumed to be in the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>- <strong>absolute</strong> <em>(for example, &quot;/projects/BuildProject/credentials/cred1&quot;)</em>—The credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>Argument type</td>
<td>String</td>
</tr>
</tbody>
</table>
# Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt;</code> ... <code>&lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code>. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource to use as the default for steps run by procedures in this project. Argument type: String</td>
</tr>
<tr>
<td>tracked</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of a workspace to use as the default for steps run by procedures in this project. Argument type: String</td>
</tr>
</tbody>
</table>

## Positional arguments

**projectName**

## Response

None or a status OK message.

### ec-perl

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

**Example**

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

### ectool

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

**Example**

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

Back to Top

# deleteProject

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

You must specify a `projectName`.

Back to Top
**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique within the project. Argument type: String</td>
</tr>
<tr>
<td>foreground</td>
<td>(Optional) &lt;Boolean flag&gt; 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

- projectName

**Response**

None or a status OK message.

**ec-perl**

**syntax:** $cmdr->deleteProject(<projectName>, {<optionals>});

**Example**

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

**ectool**

**syntax:** ectool deleteProject <projectName> ...

**Example**

ectool deleteProject "Test Proj"

Back to Top

---

**getProject**

Finds a project by its name.

You must specify a **projectName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique within the project. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- projectName

**Response**

One **project** element.
**getProjects**

Retrieves all projects.

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

**Positional arguments**

None

**Response**

Zero or more 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`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>projectName</strong></td>
<td>The name of the project that must be unique within the project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>credentialName</strong></td>
<td>Name of the credential in one of these forms:</td>
</tr>
<tr>
<td></td>
<td>- relative (for example, &quot;cred1&quot;)—The credential is assumed to be in the</td>
</tr>
<tr>
<td></td>
<td>project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>- absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The</td>
</tr>
<tr>
<td></td>
<td>credential can be from any specified project, regardless of the target</td>
</tr>
<tr>
<td></td>
<td>object’s project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>description</strong></td>
<td>A plain text or HTML description for this object.</td>
</tr>
<tr>
<td></td>
<td>If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags.</td>
</tr>
<tr>
<td></td>
<td>The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;i&gt; </code>&lt;li&gt;<code> </code>&lt;ol&gt;<code> </code>&lt;p&gt;<code> </code>&lt;pre&gt;<code> </code>&lt;span&gt;<code> </code>&lt;style&gt;<code> </code>&lt;table&gt;<code> </code>&lt;tc&gt;<code> </code>&lt;td&gt;<code> </code>&lt;th&gt;<code> </code>&lt;tr&gt;<code> </code>&lt;ul&gt;`</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>newName</strong></td>
<td>New name of the project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>resourceName</strong></td>
<td>The name of the resource used as the default for steps run by procedures in</td>
</tr>
<tr>
<td></td>
<td>this project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>tracked</strong></td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>If set to 1 or true, Change Tracking is enabled for the project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td><strong>workspaceName</strong></td>
<td>The name of the default workspace where job output is stored.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `projectName`

**Response**

None or a status OK message.
ec-perl

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

**Example**

$cmdr->modifyProject("Test Proj", {description => "A very simple project");

To enable Change Tracking for the Default project:

$cmdr->modifyProject("Default", {tracked => true});

ectool

**syntax:** ectool modifyProject <projectName> ...

**Example**

ectool modifyProject "Test Proj" --description "A very simple project"

To enable Change Tracking for the Default project:

ectool modifyProject "Default" --tracked true

reloadSetupScripts

Runs new, modified, or previously unsuccessful setup scripts.

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

**Positional arguments**

None

**Response**

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 502
- `deleteProperty` on page 508
- `expandString` on page 513
- `getProperties` on page 518
- `getProperty`
- `incrementProperty`
- `modifyProperty`
- `setProperty`

**createProperty**

Creates a regular string or nested property sheet using a combination of property path and context.

You must specify a `propertyName` and locator arguments to define where (or on which object) you are creating this property.

**Note:** The names "properties" and "project" are not valid property names.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>propertyName</code></td>
<td>The name of the property that must be unique within the property sheet.</td>
</tr>
<tr>
<td></td>
<td>It can be a relative or absolute property path, including &quot;my&quot; paths such as &quot;/myProject/prop1&quot;.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>(Optional) The name of the application container of the property sheet which owns the property. The name must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>applicationTierName</code></td>
<td>(Optional) The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>artifactName</code></td>
<td>(Optional) The name of the artifact container of the property sheet which owns the property.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong>: An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly.</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration container that owns the property.</td>
</tr>
<tr>
<td>counter</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>If 1 or true, the property is used as a counter.</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential container of the property sheet which owns the property.</td>
</tr>
<tr>
<td></td>
<td>Name of the credential in one of these forms:</td>
</tr>
<tr>
<td></td>
<td>- relative (for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>- absolute (for example, &quot;projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
<tr>
<td>credentialProtected</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code>.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment container of the property sheet that owns the property. The name must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) The name of the environment template container of the property sheet that owns the property.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) The name of the environment template tier container of the property sheet that owns the property.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier container of the property sheet that owns the property.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>expandable</td>
<td>(Optional) Whether or not the property is recursively expandable. `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>This determines whether the property value will undergo property expansion when it is fetched. The default is true.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>extendedContextSearch</td>
<td>(Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway container of the property sheet.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of the group group container of the property sheet which owns the property.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobId</td>
<td>(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.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier.</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) The object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) Property path string.</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin container of the property sheet which owns the property. When using this argument, you must also enter <code>projectName</code>.</td>
</tr>
<tr>
<td>procedureName</td>
<td>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>.</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the process when the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project container of the property sheet which owns the property. The name must be unique among all projects.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>propertyType</td>
<td>(Optional) Type of property: `&lt;string</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository container of the property sheet which owns the property. Use the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(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</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of the resource pool (with one or more resources) container of the property sheet that owns the property. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) The name of the resource template container of the property sheet that owns the property. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(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</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state container of the property sheet which owns the property Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step container of the property sheet which owns the property. If you use a step name to define the location for the new property, you must also enter projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) The name of the special system object. In this context, only server is legal. Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>value</td>
<td>(Optional) For a string property (see propertyType), this is the value of the property. For a sheet property, this argument is invalid. Argument type: String</td>
</tr>
<tr>
<td>valueFile</td>
<td>(Optional) This option is supported only in Perl and ectool bindings - it is not a part of the XML protocol. The contents of the valueFile is read and stored in the &quot;value&quot; field for a string property. This is an alternative argument for value and is useful if the &quot;value&quot; field spans multiple lines.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace container of the property sheet. Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone container of the property sheet. Argument type: String</td>
</tr>
</tbody>
</table>

**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**

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

**ectool**

*Syntax:* `ectool createProperty <propertyName> ...`

**Examples**

```ectool
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.

**Note**: The names "properties" and "project" are not valid property names.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| `propertyName`  | The name of the property that must be unique within the property sheet.  
|                 | Argument type: String                                                                                                                      |
| `applicationName` | (Optional) The name of the application that must be unique among all projects.  
|                 | Argument type: String                                                                                                                      |
| `applicationTierName` | (Optional) The name of the application tier.  
|                 | Argument type: String                                                                                                                      |
| `artifactName`  | (Optional) The name of the artifact.  
|                 | Argument type: String                                                                                                                      |
| `artifactVersionName` | (Optional) The name of the artifact version.  
|                 | Note: An artifact version name is interpreted by the server as the `artifactVersionName` attribute for the `artifactVersion` in question. This name is parsed and interpreted as "groupId:artifactKey:version" and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly.  
|                 | Argument type: String                                                                                                                      |
| `componentName` | (Optional) The name of the component.  
|                 | Argument type: String                                                                                                                      |
| `configName`    | (Optional) The name of the email configuration.  
|                 | Argument type: String                                                                                                                      |
| `credentialName` | (Optional) Whether or not the property is recursively expandable. Name of the credential in one of these forms:  
|                 | - relative (for example, "cred1")–The credential is assumed to be in the project that contains the request target object.  
|                 | - absolute (for example, "/projects/BuildProject/credentials/cred1")–The credential can be from any specified project, regardless of the target object’s project.  
<p>|                 | Argument type: String                                                                                                                      |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) The name of the environment template. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) The name of the environment template tier. Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier. Argument type: String</td>
</tr>
</tbody>
</table>
| extendedContextSearch     | (Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property.  
Optional flag -0|1|true|false If set, and there is an object specifier in the command, ElectricFlow first looks for the property in that object specifier, but also searches in other locations if not found, according to the following rules:  
1. If the object specifier is a procedure, ElectricFlow looks for the property in the project where the procedure resides.  
2. If the object specifier is a job step, ElectricFlow looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties.  
The default setting is true. Argument type: Boolean |
| gatewayName               | (Optional) The name of the gateway. Argument type: String                                                                                     |
| groupName                 | (Optional) The name of a group that contains this property. Argument type: String                                                               |
| jobId                     | (Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created.  
The system also accepts a job name assigned to the job by its name template. Argument type: UUID |
<p>| jobStepId                 | (Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID               |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) This is an object identifier returned by findObjects and getObjects. Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) Property path. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of a plugin that may contain a property you want to delete. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure containing the property you want to delete. Also requires projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project that contains the property you want to delete. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet, assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource that contains the property you want to delete.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) The name of the resource template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule containing the property you want to delete.</td>
</tr>
<tr>
<td></td>
<td><strong>Also requires</strong> projectName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step containing the property you want to delete.</td>
</tr>
<tr>
<td></td>
<td><strong>Also requires</strong> projectName and procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) The name of a special system object. Only &quot;server&quot; is valid in this context.</td>
</tr>
<tr>
<td></td>
<td>Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition.</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition.</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The user name that contains this property.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone.</td>
</tr>
</tbody>
</table>

**Positional arguments**

propertyName

**Response**

None or a status OK message.

**ec-perl**

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

*Example*

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

**ectool**

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

*Example*

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

**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.

**Note:** The names "properties" and "project" are not valid property names.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>The string value to expand in the given context. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier. Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact. Argument type: String</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(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 &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component. Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential container of the property sheet which owns the property. Name of the credential in one of these forms: relative (for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the target object’s project. Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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                                                       |
| gatewayName       | (Optional) The name of the gateway.  
                          Argument type: String                                                       |
| groupName         | (Optional) The name of a group where you might expand a string.  
                          Argument type: String                                                       |
| jobId             | (Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created.  
                          The system also accepts a job name assigned to the job by its name template.  
                          Argument type: UUID                                                         |
| jobStepId         | (Optional) The unique identifier for a job step that is assigned automatically when the job step is created.  
                          Argument type: UUID                                                         |
| notifierName      | (Optional) The name of the email notifier.  
                          Argument type: String                                                       |
| objectId          | (Optional) This is an object identifier returned by findObjects and getObjects.  
                          Argument type: String                                                       |
| path              | (Optional) Property path string.  
                          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                                                       |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>(Optional) The name of a procedure where you might need to expand a string. <strong>Also requires</strong> projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project that contains the string to expand. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is automatically assigned when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of a resource where you might expand a string. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of a schedule within this project. <strong>Also requires</strong> projectName. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step whose string you might be expanding. Also requires projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) System object names include: admin</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user where you may need to expand the string. Argument type: String</td>
</tr>
<tr>
<td>valueFile</td>
<td>(Optional) This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the valuefile is read and stored in the &quot;value&quot; field. This is an alternative argument for value and is useful if the value field spans multiple lines.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
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*

```perl
$cmdr->expandString('$[fullUserName]', {userName => "admin"})->findvalue('//value')
```  
```perl
$cmdr->expandString('$[/myWorkspace/agentUncPath]/$[logFileName]',
  {jobStepId => 5da765dd-73f1-11e3-b67e-b0a420524153})->findvalue('//value')
```  

** ectool  

*Syntax:* ectool expandString <value> ...

*Examples*

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

**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.

*Note:* The names "properties" and "project" are not valid property names.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier. Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact. Argument type: String</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component. Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration. Argument type: String</td>
</tr>
</tbody>
</table>
| credentialName          | (Optional) The name of the credential containing the properties to retrieve. Name of the credential in one of these forms:  
  - **relative** (for example, "cred1")–The credential is assumed to be in the project that contains the request target object.  
  - **absolute** (for example, "/projects/BuildProject/credentials/cred1")–The credential can be from any specified project, regardless of the target object's project. Argument type: String |
<p>| 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                                                                                   |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>expand</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>The default value is 1 (true), and the value of each property will be expanded.</td>
</tr>
<tr>
<td></td>
<td>A value of 0 (false) will cause the unexpanded value of each property to be returned.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of the group containing the properties to retrieve.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code>.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin containing the properties to retrieve. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the properties to retrieve. Also requires projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project containing the properties to retrieve. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>recurse</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource containing the properties to retrieve.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule containing the properties to retrieve.</td>
</tr>
<tr>
<td></td>
<td>Also requires projectName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step containing the properties to retrieve.</td>
</tr>
<tr>
<td></td>
<td>Also requires projectName and procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) The name of the system object containing the properties to</td>
</tr>
<tr>
<td></td>
<td>retrieve. Only &quot;server&quot; is supported.</td>
</tr>
<tr>
<td></td>
<td>Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user containing the properties to retrieve.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow. Also requires projectName. Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace containing the properties to retrieve.</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone. Argument type: String</td>
</tr>
</tbody>
</table>

### 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:**

```perl
$cdr->getProperties({<optionals>});
```

**Examples**

```perl
$cdr->getProperties({resourceName => "r2");
```

#### ectool

**syntax:**

```shell
ectool getProperties ...
```

**Examples**

```shell
ectool getProperties --resourceName "r2"
```

[Back to Top]

### getPropertyValue

Retrieves the specified property value.
You must specify a `propertyName`.

**Note:** This specification can be the full path to the property or it can be relative to an object, which then requires appropriate object locators.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>propertyName</code></td>
<td>The name or path for the property to retrieve.</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>(Optional) The name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td><code>applicationTierName</code></td>
<td>(Optional) The name of the application tier.</td>
</tr>
<tr>
<td><code>artifactName</code></td>
<td>(Optional) The name of the artifact.</td>
</tr>
<tr>
<td><code>artifactVersionName</code></td>
<td>(Optional) The name of the artifact version.</td>
</tr>
<tr>
<td><code>componentName</code></td>
<td>(Optional) The name of the component.</td>
</tr>
<tr>
<td><code>configName</code></td>
<td>(Optional) The name of the email configuration.</td>
</tr>
<tr>
<td><code>credentialName</code></td>
<td>(Optional) The name of the credential containing the property to retrieve.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment container that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>expand</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>extendedContextSearch</td>
<td>(Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property.</td>
</tr>
<tr>
<td></td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>• If the object locator is a procedure, ElectricFlow looks for the property in the project where the procedure resides.</td>
</tr>
<tr>
<td></td>
<td>• If the object locator is a job step, ElectricFlow looks in the actual parameters of the job to which it belongs, and then looks at the job properties.</td>
</tr>
<tr>
<td></td>
<td>Default setting is true.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of the group containing the property to retrieve.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobId</td>
<td>(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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: String</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) This is an object identifier returned by findObjects and getObjects. Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(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</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin containing the property to retrieve. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure containing the property to retrieve. Also requires projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| projectName            | (Optional) The name of the project containing the property to retrieve.  
Argument type: String                                                                                                                                                                                                                                                                                                                     |
| propertySheetId       | (Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created.  
Argument type: UUID                                                                                                                                                                                                                                                                                                                     |
| releaseName            | (Optional) The name of the release container of the property sheet which owns the property.  
Argument type: String                                                                                                                                                                                                                                                                                                                     |
| repositoryName         | (Optional) The name of the repository for artifact management.  
Argument type: String                                                                                                                                                                                                                                                                                                                     |
| resourceName           | (Optional) The name of the resource containing the property to retrieve.  
Argument type: String                                                                                                                                                                                                                                                                                                                     |
| resourcePoolName       | (Optional) The name of a pool containing one or more resources.  
Argument type: String                                                                                                                                                                                                                                                                                                                     |
| resourceTemplateName  | (Optional) Name of the resource template.  
Argument type: String                                                                                                                                                                                                                                                                                                                     |
| scheduleName           | (Optional) The name of the schedule containing the property to retrieve.  
**Also requires** projectName.  
Argument type: String                                                                                                                                                                                                                                                                                                                     |
| 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                                                                                                                                                                                                                                                                                       |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>systemObjectName</td>
<td>(Optional) The name of the system object containing the property to retrieve. Only &quot;server&quot; is supported.</td>
</tr>
<tr>
<td></td>
<td>Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user containing the property to retrieve.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace containing the property to retrieve.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

propertyName

**Response**

A *property sheet* or a text string containing the value of the property.

Property value example: 35491

**ec-perl**

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

**Examples**

use XML::XPath;

```
$cmdr-getProperty("/myProject/changeset ID")-findvalue('//value')-value();
```
getProperty

```
$cmdr->getProperty("Changeset ID", {projectName => "Sample Project"})
```

**ectool**

**syntax:** ectool getProperty <propertyName> ...

**Examples**

ectool getProperty "/myProject/changeset ID"

ectool getProperty "Changeset ID" --projectName "Sample Project"

# Retrieve the /users/<userName>/providerName property.

ectool getProperty --objectId <ID> --propertyName "/users/<userName>/providerName"

**incrementProperty**

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.

**Note:** The names "properties" and "project" are not valid property names.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>Name for the property that must be unique within the property sheet.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>incrementBy</td>
<td>This is positive or negative integer.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component. Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) Name of the credential in one of these forms: <strong>relative</strong> (for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the request target object. <strong>absolute</strong> (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the target object’s project. Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) Name of the environment template. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier. Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| extendedContextSearch | (Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property.  
  `<Boolean flag - 0|1|true|false>`  
  If set, and there is an object specified in the command, ElectricFlow first looks for the property in that object specifier, but also searches in other locations if not found, according to the following rules:  
  - If the object specifier is a procedure, ElectricFlow looks for the property in the project where the procedure resides.  
  - If the object specifier is a job step, ElectricFlow looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties.  
  The default setting is true.  
  Argument type: Boolean |
| 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 |
| jobId            | (Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template.  
  Argument type: UUID |
| jobStepId        | (Optional) The unique identifier for a job step that is assigned automatically when the job step is created.  
  Argument type: UUID |
| notifierName     | (Optional) The name of the email notifier.  
  Argument type: String |
| objectId         | (Optional) This is an object identifier returned by `findObjects` and `getObjects`.  
  Argument type: String |
| path             | (Optional) Path to the property.  
  Argument type: String |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin containing a property to increment.</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure containing this property. <strong>Also requires</strong> projectName. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing this property. <strong>Also requires</strong> procedureName. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource containing this property. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>Name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule containing this property. <strong>Also requires</strong> <code>projectName</code>. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step containing this property. <strong>Also requires</strong> <code>projectName</code> and <code>procedureName</code>. Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) Only server is a valid system object for this API. Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user containing this property. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace containing this property. Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

propertyName, incrementBy

**Response**

A text string containing the updated numeric property value.

**ec-perl**

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

**Examples**

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

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

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

**ectool**

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

**Examples**

ectool incrementProperty "Build Number" 1 --procedureName "Delay" --projectName "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.

**Note:** The names "properties" and "project" are not valid property names.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>The name of the property that must be unique within the property sheet. This argument can be a path. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application. Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) The name of the artifact. Argument type: String</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>(Optional) The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the artifactVersionName attribute for the artifactVersion in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name—the ElectricFlow server interprets either name form correctly. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component. Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration. Argument type: String</td>
</tr>
<tr>
<td>counter</td>
<td>(Optional) **&lt;Boolean flag -0</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) <strong>credentialName can be one of two forms:</strong> relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| credentialProtected      | *(Optional) <Boolean flag - 0|1|true|false>*  
If 1 or true, permissions required for modify privileges on the property sheet will also require execute privileges on the credentials attached to the property sheet owner.  
Argument type: Boolean |
| description              | *(Optional) A plain text or HTML description for this object.*  
If using HTML, you must surround your text with <html> ... </html> tags. The only HTML tags allowed in the text are:  
<a> <b> <br> <div> <dl> <i> <li> <ol> <p> <pre> <span> <style> <table> <tc> <td> <th> <tr> <ul>  
Argument type: String |
| environmentName          | *(Optional)* The name of the environment that must be unique among all projects.  
Argument type: String |
| environmentTemplateName  | *(Optional)* Name of the environment template.  
Argument type: String |
| environmentTemplateTierName | *(Optional)* Name of the environment template tier.  
Argument type: String |
| environmentTierName      | *(Optional)* The name of the environment tier.  
Argument type: String |
| expandable               | *(Optional) <Boolean flag - 0|1|true|false>*  
Determines whether the property will undergo property expansion when it is retrieved.  
The default is true.  
Argument type: Boolean |
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| extendedContextSearch | (Optional) For simple property names, whether or not to search objects in the hierarchy to find the desired property.  
  \<Boolean flag \> 0|1|true|false– If set, and there is an object specified in the command, ElectricFlow first looks for the property in that object specifier, but also searches in other locations if not found, according to the following rules:  
  1. If the object specifier is a procedure, ElectricFlow looks for the property in the project where the procedure resides.  
  2. If the object specifier is a job step, ElectricFlow looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties.  
  The default setting is true.  
  Argument type: Boolean |
| gatewayName       | (Optional) The name of the gateway.  
  Argument type: String |
| groupName         | (Optional) The name of the group.  
  Argument type: String |
| jobId             | (Optional) The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template.  
  Argument type: UUID |
| jobStepId         | (Optional) The unique identifier for a job step, 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 |
| objectId          | (Optional) An object identifier returned by findObjects and getObjects.  
  Argument type: String |
| path              | (Optional) Path to the property.  
  Argument type: String |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) Name of the plugin. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) Name of the procedure. <strong>Also requires</strong> projectName. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(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</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) Name of the process when the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) Name of the process step when the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet, which is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>propertyType</td>
<td>(Optional) `&lt;string</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) Name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) Name of the resource. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) Name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) Name of the schedule. Also requires projectName. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) Name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) Name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>Name of the step containing the property to be modified. Also requires projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>System objects include: admin</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user. Argument type: String</td>
</tr>
<tr>
<td>value</td>
<td>(Optional) Value of the property. Argument type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
valueFile | (Optional) This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. The contents of the `valueFile` is read and stored in the "value" field. This is an alternative argument for `value` and is useful if the "value" field spans multiple lines.
workflowDefinitionName | (Optional) 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

`propertyName`

Response

An XML stream that echoes the modified property.

**ec-perl**

*Syntax:* `$cmdr->modifyProperty(<propertyName>, {...});`

*Example*

```
$cmdr->modifyProperty("Saved Variables", {description => "Starting configuration of name/value pairs", jobId => 4fa765dd-73f1-11e3-b67e-b0a420524153});
```

**ectool**

*Syntax:* `ectool modifyProperty <propertyName> ...

*Example*

```
ectool modifyProperty "Saved Variables" --description "Starting configuration of name/value pairs" --jobId 4fa765dd-73f1-11e3-b67e-b0a420524153
```

Back to Top

**setProperty**

Sets the value for the specified property.
You must specify 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.

**Note:** The names "properties" and "project" are not valid property names.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>propertyName</code></td>
<td>The name or path of the property that must be unique within the property sheet. This argument can be a path. Argument type: String</td>
</tr>
<tr>
<td><code>value</code></td>
<td>The value of the property. Argument type: String</td>
</tr>
<tr>
<td><code>valueFile</code></td>
<td>This option is supported only in Perl and ectool bindings - it is not part of the XML protocol. Contents of the <code>valueFile</code> are read and stored in the &quot;value&quot; field. This is an alternative argument for <code>value</code> and is useful if the value field spans multiple lines.</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>(Optional) The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td><code>applicationTierName</code></td>
<td>(Optional) The name of the application tier. Argument type: String</td>
</tr>
<tr>
<td><code>artifactName</code></td>
<td>(Optional) The name of the artifact. Argument type: String</td>
</tr>
<tr>
<td><code>artifactVersionName</code></td>
<td>(Optional) The name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly. Argument type: String</td>
</tr>
<tr>
<td><code>componentName</code></td>
<td>(Optional) The name of the component. Argument type: String</td>
</tr>
<tr>
<td><code>configName</code></td>
<td>(Optional) The name of the email configuration. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential containing the property you want to set. Name of the credential in one of these forms:</td>
</tr>
<tr>
<td></td>
<td>- relative (for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the request target object.</td>
</tr>
<tr>
<td></td>
<td>- absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the target</td>
</tr>
<tr>
<td></td>
<td>object's project.</td>
</tr>
<tr>
<td></td>
<td>Also requires projectName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only</td>
</tr>
<tr>
<td></td>
<td>text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;font&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) Name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) The name of the environment tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>expandable</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>The default is 1 (true), and the property value will be expanded when referenced.</td>
</tr>
<tr>
<td></td>
<td>If you do not want the property to expand, use the value of 0 (false).</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
</tbody>
</table>
**Arguments** | **Descriptions**
--- | ---
**extendedContextSearch** | (Optional) `<Boolean flag - 0|1|true|false>
If set, and there is an object specified in the command, ElectricFlow first looks for the property in the object specified, but also searches in other locations if not found, according to the following rules:
If the object specified is a procedure, ElectricFlow looks for the property in the project where the procedure resides.
If the object specified is a job step, ElectricFlow looks in the actual parameters of the procedure to which it belongs, and then looks at the job properties.
The default setting is false.
Argument type: Boolean

**gatewayName** | (Optional) The name of the gateway containing the property that you want to set.
Argument type: String

**groupName** | (Optional) The name of the group containing the property you want to set.
Argument type: String

**jobId** | (Optional) The name of the job containing the property you want to set. The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template.
Argument type: UUID

**jobStepId** | (Optional) The name of the job step containing the property that you want to set. The unique identifier for a job step, assigned automatically when the job step is created.
Argument type: UUID

**objectId** | (Optional) This is an object identifier returned by `findObjects` and `getObjects`.
Argument type: String

**notifierName** | (Optional) The name of the email notifier.
Argument type: String

**path** | (Optional) The path to the property.
Argument type: String
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin containing the property you want to set.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure containing the property that you want to set.</td>
</tr>
<tr>
<td></td>
<td>Also requires projectName. (Optional) Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process when the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step when the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project containing the property that you want to set. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: String</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource containing the property that you want to set. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td><strong>Descriptions</strong></td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule containing the property you want to set. <strong>Also requires</strong> projectName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) The name of the state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step containing the property you want to set. <strong>Also requires</strong> projectName and procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) The name of the system object containing the property you want to set. System objects include: admin</td>
</tr>
<tr>
<td></td>
<td>Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user containing the property you want to set.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace containing the property that you want to set.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**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> ...

*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"
API Commands - Resource Management

- addResourcesToPool on page 547
- addResourceToEnvironmentTier on page 548
- createResource on page 549
- createResourcePool on page 552
- deleteResource on page 554
- deleteResourcePool on page 554
- getResource on page 555
- getResources on page 556
- API Commands - Resource Management on page 547
- getResourcesInPool on page 557
- getResourcePool on page 558
- getResourcePools on page 559
- getResourceUsage on page 559
- modifyResource on page 560
- pingAllResources on page 564
- pingResource on page 565
- removeResourceFromEnvironmentTier on page 566
- removeResourcesFromPool on page 567
- signCertificate on page 568

**addResourcesToPool**

Adds resources to a specific resource pool. A resource pool is a named group of resources. You must specify a resourcePoolName and one or more resource names.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourcePoolName</td>
<td>Name for the resource pool that must be unique among all resource pools.</td>
</tr>
<tr>
<td></td>
<td>The resource pool has one or more resources.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>resourceNames</td>
<td>List of resources to add to the resource pool.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Collection</td>
</tr>
</tbody>
</table>
Positional arguments
  resourceName, resourceNames

Response
  None or status OK message.

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

Example
  $cmdr->addResourcesToPool("pool1", { resourceName => ["resoucre1", "resource2", "resource3"]});

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

Example
  ectool addResourcesToPool "Test Pool" --resourceNames Test1 Test2 Test3

addResourceToEnvironmentTier

Adds a resource to the specified environment tier.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name for the resource that must be unique among all resources.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>Name of the environment that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>Name for the environment tier that must be unique among all tiers for the environment.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Positional arguments
  resourceName, projectName, environmentName, environmentTierName
Response
None or a status OK message.

dc-perl
Syntax:
$<object>-addResourceToEnvironmentTier(<resourceName>, <projectName>,
<environmentName>, <environmentTierName>);

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

dcctool
Syntax:
addResourceToEnvironmentTier <resourceName> <projectName> <environmentName> <environmentTierName>

Example:
dcctool addResourceToEnvironmentTier Resource1 default newEnv envTier1

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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name of the new resource that must be unique among all resources.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>artifactCacheDirectory</td>
<td>(Optional) Directory on the agent host where retrieved artifacts are stored.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>block</td>
<td>(Optional) &lt;Boolean flag&gt; - 0/1/true/false</td>
</tr>
<tr>
<td></td>
<td>A newly created resource will be pinged. This argument makes the createResource call block until the result of the ping is known. The default is false.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Boolean</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt;</code> ... <code>&lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code>. Argument Type: String</td>
</tr>
<tr>
<td>hostName</td>
<td>(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</td>
</tr>
<tr>
<td>hostType</td>
<td>(Optional) Type of the host. Argument Type: String</td>
</tr>
<tr>
<td>pools</td>
<td>(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</td>
</tr>
<tr>
<td>port</td>
<td>(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 &quot;Server Settings&quot; page in the automation platform UI. For ssh, the default is 22. Argument Type: Integer</td>
</tr>
<tr>
<td>proxyCustomization</td>
<td>(Optional) Customized Perl code specifying how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources. Argument Type: String</td>
</tr>
<tr>
<td>proxyHostName</td>
<td>(Optional) The name or IP address of the computer containing the ElectricFlow Agent used for a proxy resource. Argument Type: String</td>
</tr>
<tr>
<td>proxyPort</td>
<td>(Optional) The ElectricFlow agent port number for a proxy resource. See the port argument description for more details. Argument Type: Integer</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>proxyProtocol</td>
<td>(Optional) Protocol for communicating with the proxy target. Defaults to ssh. (This argument is not exposed in the ElectricFlow UI at this time.)</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>repositoryNames</td>
<td>(Optional) A list of one or more repository names. Each repository name is listed on a &quot;new line.&quot;</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>resourceDisabled</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>If this is set to 1, ElectricFlow will not start new steps on this resource. The default is <code>false</code>.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Boolean</td>
</tr>
<tr>
<td>shell</td>
<td>(Optional) This sets a default shell for running step commands on this resource. The default is &quot;cmd /q /c&quot; for a Windows agent and &quot;sh -e&quot; for a UNIX agent.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>stepLimit</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Integer</td>
</tr>
<tr>
<td>trusted</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>If this is set to <code>true</code>, the resource is <code>trusted</code>.</td>
</tr>
<tr>
<td></td>
<td>Agents can be either <code>trusted</code> or <code>untrusted</code>:</td>
</tr>
<tr>
<td></td>
<td>• <code>trusted</code>--The ElectricFlow server verifies the agent’s identity using SSL certificate verification.</td>
</tr>
<tr>
<td></td>
<td>• <code>untrusted</code>--The ElectricFlow server does not verify agent identity. An untrusted agent could be a security risk.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Boolean</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>useSSL</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) Name of the workspace that the resource uses. Argument Type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone where this resource resides. Argument Type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

```
resourceName
```

### Response

None or a status OK message.

#### ec-perl

**syntax:**
```
$cmdr->createResource(<resourceName>, [<optionals>]);
```

**Example**
```
$cmdr->createResource("Test Resource 1", {hostName => "localhost", pools => "P1 P2");
```

#### ectool

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

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

### createResourcePool

Creates a new pool for resources.

You must specify a `resourcePoolName`. 

---

**ElectricFlow**

552
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourcePoolName</td>
<td>Name for the resource pool that must be unique among all resource pools. Argument Type: String</td>
</tr>
<tr>
<td>autoDelete</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt; Argument Type: String</td>
</tr>
<tr>
<td>orderingFilter</td>
<td>A Javascript block invoked when scheduling resources for a pool. Note: A Javascript block is not required unless you need to override the default resource ordering behavior. Argument Type: String</td>
</tr>
<tr>
<td>resourceNames</td>
<td>A list of resource names to add to the pool. This value does not need to refer to an existing resource. Any names that do not resolve to an existing resource will be skipped when assigning resources to steps. Argument Type: Collection</td>
</tr>
<tr>
<td>resourcePoolDisabled</td>
<td>&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

### Positional arguments

- resourcePoolName

### Response

Returns a resourcePool object.

### ec-perl

**syntax:** $cmdr->createResourcePool(<resourcePoolName>, {<optionals>});

**Example**

```
$cmdr->createResourcePool("aPool", {resourceName => ["resource1", "resource2"]});
```
**ectool**  

* syntax:  
  `ectool createResourcePool <resourcePoolName> ...`

* Example  
  `ectool createResourcePool aPool --resourceNames resource1 resource2`

**deleteResource**

Deletes a resource.  
You must enter a resourceName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name of the resource that must be unique among all resources. Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

resourceName

**Response**

None or a status OK message.

**ec-perl**

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

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

**ectool**

* syntax:  
  `ectool deleteResource <resourceName>`

* Example  
  `ectool deleteResource "Test Resource 1"`

**deleteResourcePool**

Deletes a resource pool.  
You must enter a resourcePoolName.
**Arguments**

| resourcePoolName       | Name for the resource pool that must be unique among all resource pools. A resource pool contains one or more resources. Argument Type: String |

**Positional arguments**

resourcePoolName

**Response**

None or a status OK message.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* `ectool deleteResourcePool <resourcePoolName>`

*Example*

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

Back to Top

---

**getResource**

Retrieves a resource by its name.

You must specify `resourceName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name for the resource that must be unique among all resources. Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

resourceName

**Response**

One `resource` element, which includes the resource ID, name, agent state, time created, host name, owner, port, disabled flag, shell, step limit, workspace name, and so on. If using zones and gateways, getResource returns a list of gateways where this resource participates.

**ec-perl**

*Syntax:* `$cmdr->getResource(<resourceName>);`
Example

```bash
$qcmdr->getResource("Test Resource 1");
```

tool

.syntax: `ectool getResource <resourceName>`

Example

```bash
ectool getResource "Test Resource 1"
```

getResources

Retrieves all resources.

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

Positional arguments

None

Response

Zero or more resource elements.

tool

.syntax: `ectool getResources()`

Example

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

tool

.syntax: `ectool getResources`

Example

```bash
ectool getResources
```

getResourcesInEnvironmentTier

Returns the list of resources in an environment tier.

You must specify the `projectName`, `environmentName` and `environmentTierName` arguments.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>Name of the environment that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>Name for the environment tier that must be unique among all tiers</td>
</tr>
<tr>
<td></td>
<td>for the environment.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

environmentName, environmentTierName

**Response**

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

**ec-perl**

**Syntax:**

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

**Example:**

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

**ectool**

**Syntax:**

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

**Example:**

```bash
ectool getResourcesInEnvironmentTier default newEnv envTier1
```

**getResourcesInPool**

Retrieves a list of resources in a pool.

You must specify a resourcePoolName.
Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourcePoolName</td>
<td>The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: UUID</td>
</tr>
</tbody>
</table>

Positional arguments

- resourcePoolName

Response

- An XML stream containing zero or more resource elements.

**ec-perl**

**syntax:**

```
$cmdr->getResourcesInPool(<resourcePoolName>, {<optionals>});
```

**Example**

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

**ectool**

**syntax:**

```
ectool getResourcesInPool <pool> ...
```

**Example**

```
ectool getResourcesInPool WindowsPool
```

getResourcePool

Retrieves a specified resource pool by name.

You must specify a resourcePoolName.

Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourcePoolName</td>
<td>Name for the resource pool that must be unique among all resource pools.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Positional arguments

- resourcePoolName

Response

- An XML stream containing one resourcePool element.
**ec-perl**

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

*Example*

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

**ectool**

*syntax:* ectool getResourcePool <resourcePoolName>

*Example*

ectool getResourcePool WindowsPool

**get(ResourcePools)**

Retrieves a list of resource pools.

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

**Positional arguments**

None

**Response**

An XML stream containing zero or more resourcePool elements.

**ec-perl**

*syntax:* \$cmdr->getResourcePools;

*Example*

\$cmdr->getResourcePools;

**ectool**

*syntax:* ectool getResourcePools

*Example*

ectool getResourcePools

**get(ResourceUsage)**

Retrieves resource usage information.
## 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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name for the resource that must be unique among all resources. Argument Type: String</td>
</tr>
<tr>
<td>artifactCacheDirectory</td>
<td>The directory on the agent host where retrieved artifacts are stored. Argument Type: String</td>
</tr>
<tr>
<td>block</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code> Argument Type: String</td>
</tr>
<tr>
<td>hostName</td>
<td>The name or IP address for the ElectricFlow machine containing the agent for this resource.                                                                                                                 Argument Type: String</td>
</tr>
<tr>
<td>newName</td>
<td>Enter any name of your choice to rename the resource.                                                                                                                                                        Argument Type: String</td>
</tr>
<tr>
<td>pools</td>
<td>A space-separated list of one or more pool names where this resource is a member. The pool name can be used in place of a single resource name. ElectricFlow chooses a resource from the pool when it executes the job step. Argument Type: String</td>
</tr>
<tr>
<td>port</td>
<td>The port number for the ElectricFlow agent. Default is to the default agent port, but you can change this port number because of port conflicts or multiple agents running on the same machine. Argument Type: Integer</td>
</tr>
<tr>
<td>proxyCustomization</td>
<td>Customized Perl code specifying how the proxy resource communicates with the proxy target. This applies only to proxy resources. Argument Type: String</td>
</tr>
<tr>
<td>proxyHostName</td>
<td>The IP address of the computer containing the ElectricFlow Agent used for a proxy resource. Argument Type: String</td>
</tr>
<tr>
<td>proxyPort</td>
<td>The ElectricFlow agent port number for a proxy resource. See the port argument for more details. Argument Type: Integer</td>
</tr>
<tr>
<td>proxyProtocol</td>
<td>Protocol for communicating with the proxy target. Defaults to ssh. This argument is not exposed in the ElectricFlow UI at this time. Argument Type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>repositoryNames</td>
<td>A list of repository names with each repository name listed on a &quot;new line&quot;. Argument Type: String</td>
</tr>
<tr>
<td>resourceDisabled</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>shell</td>
<td>This sets a default shell for running step commands on this resource. The default is &quot;cmd /q /c&quot; for a Windows agent and &quot;sh -e&quot; for a UNIX agent. Argument Type: String</td>
</tr>
<tr>
<td>stepLimit</td>
<td>This limits the number of steps that can be running on the resource at one time. Setting this value to 1 is a good way to enforce serial access to the resource. Argument Type: String</td>
</tr>
<tr>
<td>trusted</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>• trusted—The ElectricFlow server verifies the agent's identity using SSL certificate verification.</td>
</tr>
<tr>
<td></td>
<td>• untrusted—The ElectricFlow server does not verify agent identity. An untrusted agent could be a security risk.</td>
</tr>
<tr>
<td>useSSL</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workspaceName</td>
<td>Name of the default workspace where job output is stored. Argument Type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>Name of the zone where this resource resides, which must be unique among all zones. Argument Type: String</td>
</tr>
</tbody>
</table>
Positional arguments

**resourceName**

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* $cmdr->modifyResource(<resourceName>, {...});

*Example*

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

**ectool**

*Syntax:* ectool modifyResource <resourceName> ...

*Example*

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

**modifyResourcePool**

Modifies an existing resource pool.

You must specify a **resourcePoolName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourcePoolName</td>
<td>Name for the resource pool that must be unique among all resource pools. A resource pool contains one or more resources.</td>
</tr>
<tr>
<td>autoDelete</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with tags. The only HTML tags allowed in the text are: a b br div font i li ol p pre span style table tc td th tr ul</td>
</tr>
<tr>
<td>newName</td>
<td>Any new unique name for this resource pool.</td>
</tr>
</tbody>
</table>

**Argument Type:** String
Arguments | Descriptions
---|---
resourceNames | A list of resource names to add to the pool. This value does not need to refer to an existing resource. Any names that do not resolve to an existing resource will be skipped when assigning resources to steps.
Argument Type: Collection

orderingFilter | A Javascript block invoked when scheduling resources for a pool.

Note: A Javascript block is not required unless you need to override the default resource ordering behavior.
Argument Type: String

resourcePoolDisabled | `<Boolean flag - 0|1|true|false> - If this is set to true, any runnable steps that refer to the pool will block until the pool is enabled again.
Argument Type: Boolean

Positional arguments
resourcePoolName

Response
The modified resourcePool object.

cp-perl

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

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

ectool

Syntax: ectool modifyResourcePool <resourcePoolName> ...

Example
ectool modifyResourcePool WindowsPool --resourcePoolDisabled 1

pingAllResources
Pings all resources.
## pingAllResources

Pings all resources.

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

None or a status OK message.

### ec-perl

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

**Example**

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

### ectool

**syntax:** ectool pingAllResources...

**Example**

```ectool
ectool pingAllResources
```

## pingResource

Pings the specified resource.

You must specify the resourceName.

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name for the resource that must be unique among all resources. Argument Type: String</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name for the resource that must be unique among all resources.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>Name of the environment that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>Name for the environment tier that must be unique among all tiers for the environment.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

Response

None or a status OK message.

ec-perl

Syntax:

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

Example:
$ec->removeResourceFromEnvironmentTier("Resource1", "default", "newEnv", "envTier1");

**ectool**

**Syntax:**

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

**Example:**

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

---

**removeResourcesFromPool**

Removes resources from a specified resource pool.

You must specify a `resourcePoolName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourcePoolName</td>
<td>Name for the resource pool that must be unique among all resource pools.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>resourceNames</td>
<td>The list of resources to remove from this pool.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Collection</td>
</tr>
</tbody>
</table>

**Positional arguments**

`resourcePoolName`

**Response**

None or a status OK message.

**ec-perl**

**syntax:**

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

**Example**

```perl
$cmdr->removeResourcesFromPool("Test Pool", {resourceNames => ["Test1", "Test2", "Test3"]});
```

**ectool**

**syntax:**

```bash
ectool removeResourcesFromPool <resourcePoolName> ...
```

**Example**

```bash
ectool removeResourcesFromPool "Test Pool" --resourceNames Test1 Test2 Test3
```
**signCertificate**

Signs an agent certificate.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>certificateData</td>
<td>A Privacy Enhanced Mail (PEM) encoded certificate signing request.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- certificateData

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `$cmdr->signCertificate (<certificateData>);`

*Example*

```
$cmdr->signCertificate ("MIIEczCCAlugAwIBAgIBADANBgkqhkiG9w0BAQQFAD");
```

**ectool**

*Syntax:* `ectool signCertificate <certificateData>`

*Example*

```
ectool signCertificate "MIIEczCCAlugAwIBAgIBADANBgkqhkiG9w0BAQQFAD"
```

Back to Top

**API Commands - Schedule Management**

- **createSchedule** on page 568
- **deleteSchedule** on page 573
- **getSchedule** on page 573
- **getSchedules** on page 574
- **modifySchedule** on page 575
- **pauseScheduler** on page 579

**createSchedule**

Creates a new schedule.

**Note:** If both `startTime` and `stopTime` are specified, `intervalUnits` and `interval` are used to specify an interval time to repeat running the procedure.

You must specify a `projectName` and `scheduleName`. 
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>Name for the schedule that must be unique among all schedules for the project. Argument type: String</td>
</tr>
<tr>
<td>actualParameters</td>
<td>(Optional) Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure. Argument type: Map</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application that owns the process. Argument type: String</td>
</tr>
<tr>
<td>beginDate</td>
<td>(Optional) &lt;yyyy-mm-dd&gt; The date you want the schedule to begin. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) The name of the credential to use for user impersonation when running the procedure. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project. Argument type: String</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt; Argument type: String</td>
</tr>
<tr>
<td>endDate</td>
<td>(Optional) &lt;yyyy-mm-dd&gt; The date you want this schedule to end. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of the environment where the application runs.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) The name of the environment template used to create the dynamic environment.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierMapNa</td>
<td>(Optional) The name of the environment template tier map used to create the dynamic environment where the application runs.</td>
</tr>
<tr>
<td>me</td>
<td>Argument type: String</td>
</tr>
<tr>
<td>interval</td>
<td>(Optional) Determines the repeat interval for starting new jobs.</td>
</tr>
<tr>
<td></td>
<td>If specified, the procedure, process, or workflow will be rescheduled continiously at intervals of this length. &quot;Continuous&quot; means that the procedure, process, or workflow is rescheduled as soon as the previous job finishes.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>intervalUnits</td>
<td>(Optional) Specifies the units for the interval argument</td>
</tr>
<tr>
<td></td>
<td>&lt;hours</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>misfirePolicy</td>
<td>(Optional) &lt;ignore</td>
</tr>
<tr>
<td></td>
<td>Argument type: MisfirePolicy</td>
</tr>
<tr>
<td>monthDays</td>
<td>(Optional) Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>priority</td>
<td><em>(Optional)</em> `&lt;low</td>
</tr>
<tr>
<td>procedureName</td>
<td><em>(Optional)</em> Name of the procedure to run when the schedule is invoked. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td><em>(Optional)</em> Name of the application process to run when the schedule is invoked. Argument type: String</td>
</tr>
<tr>
<td>scheduleDisabled</td>
<td><em>(Optional)</em> `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>snapshotName</td>
<td><em>(Optional)</em> Name of the snapshot used to invoke the application process. Argument type: String</td>
</tr>
<tr>
<td>startTime</td>
<td><em>(Optional)</em> 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, <code>17:00</code>). Argument type: String</td>
</tr>
<tr>
<td>startingStateName</td>
<td><em>(Optional)</em> Name of the starting state of the workflow. Argument type: String</td>
</tr>
<tr>
<td>stopTime</td>
<td><em>(Optional)</em> 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, <code>17:00</code>). Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments and Descriptions

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tierMapName</td>
<td>(Optional) Name of the tier map that determines in which environment the application runs. Argument type: String</td>
</tr>
<tr>
<td>tierResourceCounts</td>
<td>Resource count for each resource template tier. Argument type: Map</td>
</tr>
<tr>
<td>timeZone</td>
<td>(Optional) Enter the time zone (string) you want to use for this schedule. Argument type: String</td>
</tr>
<tr>
<td>weekDays</td>
<td>(Optional) Restricts the schedule to specified days of the week. Specify days of the week separated by spaces. Use English names &quot;Monday&quot;, &quot;Tuesday&quot;, and so on. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) Name of the workflow to run when the schedule is invoked. Argument type: String</td>
</tr>
</tbody>
</table>

#### Positional arguments
- `projectName`, `scheduleName`

#### Response
- None or status OK message.

#### ec-perl
- **Syntax:**
  ```bash
  $cmdr->createSchedule(<projectName>, <scheduleName>, [<optionals>]);
  ```
- **Example**
  ```perl
  $cmdr->createSchedule('Sample Project', 'Weekend', {startTime => '00:00',
            stopTime => '23:59',
            weekDays => 'Saturday Sunday',
            interval => 1,
            intervalUnits => 'hours',
            actualParameter => [{actualParameterName => 'param1', value => 'value1'}] });
  ```

#### ectool
- **Syntax:**
  ```bash
  ectool createSchedule <projectName> <scheduleName> ...
  ```
- **Example**
  ```bash
  ectool createSchedule "Sample Project" "Weekend" --start00:00
  --stopTime 23:59 --weekDays "Saturday Sunday" --interval 1 --intervalUnits hours
  ```
deleteSchedule

Deletes a schedule.

You must specify a projectName and scheduleName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The schedule you want to delete belongs to this project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule you want to delete.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

Positional arguments
- projectName, scheduleName

Response
None or a status OK message.

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

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

ectool
syntax: ectool deleteSchedule <projectName> <scheduleName>

Example
ectool deleteSchedule "Sample Project" "Weekend"

getSchedule

Retrieves a schedule by its name.

You must specify a projectName and scheduleName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
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("Sample Project", "Build Schedule");

**ectool**

*Syntax:* ectool getSchedule <projectName> <scheduleName>

*Example*

ectool getSchedule "Sample Project" "Build Schedule"

getSchedules

Retrieves all schedules.

You must specify a projectName.

Arguments | Descriptions
---|---
projectName | Name for the project that must be unique among all projects.
| Argument type: String
applicationName | The name of the application that owns the process.
| Argument type: String
includeWorkflows | To include workflow related schedules.
| Argument type: String
processName | The name of the application process.
| Argument type: String
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tierMapName</td>
<td>The name of the tier map that determines the environment in which run the process.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

**projectName**

### Response

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

### ec-perl

**syntax:**

```perl
$cmdr->getSchedules({<projectName>} {...});
```

**Example**

```perl
$cmdr->getSchedules("Sample Project" {applicationName => "DeployApp"});
```

### ectool

**syntax:**

```bash
ectool getSchedules <projectName> ...
```

**Example**

```bash
ectool getSchedules "Sample Project" --applicationName "DeployApp"
```

---

## modifySchedule

Modifies an existing schedule.

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

**Note:** If both `startTime` and `stopTime` are specified, `intervalUnits` and `interval` are used to specify an interval to repeat running the procedure.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the schedule to modify.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule to modify.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>actualParameter</code></td>
<td>(Optional) Specifies the values to pass as parameters to the called procedure. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the called procedure. Argument type: Map</td>
</tr>
<tr>
<td><code>beginDate</code></td>
<td>(Optional) <code>&lt;yyyy-mm-dd&gt;</code> The date you want the schedule to begin. Argument type: String</td>
</tr>
<tr>
<td><code>clearActualParameters</code></td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><code>credentialName</code></td>
<td>(Optional) The name of the credential to use for user impersonation when running the procedure. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project. Argument type: String</td>
</tr>
<tr>
<td><code>description</code></td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; </code>&lt;b&gt; <code>&lt;br&gt; </code>&lt;div&gt; <code>&lt;dl&gt; </code>&lt;font&gt; <code>&lt;i&gt; </code>&lt;li&gt; <code>&lt;ol&gt; </code>&lt;p&gt; <code>&lt;pre&gt; </code>&lt;span&gt; <code>&lt;style&gt; </code>&lt;table&gt; <code>&lt;tc&gt; </code>&lt;td&gt; <code>&lt;th&gt; </code>&lt;tr&gt; `&lt;ul&gt; Argument type: String</td>
</tr>
<tr>
<td><code>endDate</code></td>
<td>(Optional) <code>&lt;yyyy-mm-dd&gt;</code> The date you want this schedule to end. Argument type: String</td>
</tr>
<tr>
<td><code>interval</code></td>
<td>(Optional) Determines the repeat interval for starting new jobs. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>intervalUnits</td>
<td>(Optional) Specifies the units for the interval argument &lt;hours</td>
</tr>
<tr>
<td></td>
<td>Argument type: IntervalUnits</td>
</tr>
<tr>
<td>misfirePolicy</td>
<td>(Optional) &lt;ignore</td>
</tr>
<tr>
<td></td>
<td>When the underlying issue is resolved, the server will schedule the next job at the next regularly scheduled time slot if the policy is 'ignore', otherwise it will run the job immediately. The default is ignore.</td>
</tr>
<tr>
<td></td>
<td>Argument type: IntervalUnits</td>
</tr>
<tr>
<td>monthDays</td>
<td>(Optional) Restricts the schedule to specified days of the month. Specify numbers from 1 to 31, separating multiple numbers with a space.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>newName</td>
<td>(Optional) New name of the schedule.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>priority</td>
<td>(Optional) &lt;low</td>
</tr>
<tr>
<td></td>
<td>Argument type: JobPriority</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure to run when the schedule is invoked.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process to run when the schedule is invoked.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>scheduleDisabled</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
</tbody>
</table>
## Arguments and Descriptions

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>snapshotName</td>
<td>(Optional) The name of the snapshot to use when the application process is invoked. Argument type: String</td>
</tr>
<tr>
<td>startingStateName</td>
<td>(Optional) The name of the starting state of the workflow. Argument type: String</td>
</tr>
<tr>
<td>startTime</td>
<td>(Optional) The time of day to begin running the procedure, process, or workflow when the schedule is invoked. Enter hours and minutes, formatted \texttt{hh:mm}, using the 24-hour clock (for example, 17:00). ElectricFlow starts creating jobs at this time on the days specified. Argument type: String</td>
</tr>
<tr>
<td>stopTime</td>
<td>(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 \texttt{stopTime} is not specified, ElectricFlow creates one job only on each specified day. Enter hours and minutes, formatted \texttt{hh:mm}, using the 24-hour clock (for example, 17:00). Argument type: String</td>
</tr>
<tr>
<td>tierMapName</td>
<td>(Optional) The name of the tier map that maps application processes to environments. Argument type: String</td>
</tr>
<tr>
<td>timeZone</td>
<td>(Optional) Enter the time zone you want to use for this schedule. Argument type: String</td>
</tr>
<tr>
<td>weekDays</td>
<td>(Optional) Restrictions the schedule to specified days of the week. Specify days of the week separated by spaces. Use English names &quot;Monday&quot;, &quot;Tuesday&quot;, and so on. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow to run when the schedule is invoked. Argument type: String</td>
</tr>
</tbody>
</table>

### Positional arguments

- projectName, scheduleName

### Response

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

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

*Example*

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

**ectool**

*Syntax:* 
```bash
ectool modifySchedule <projectName> <scheduleName> ...
```

*Example*

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

---

**pauseScheduler**

Sets the scheduler to pause.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>paused</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>If this argument set to 1 or true, set the scheduler to pause.</td>
</tr>
<tr>
<td></td>
<td>If this argument is set to 0 or false, set the scheduler to not pause.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`paused`

**Response**

None or a status OK message.

---

**ec-perl**

*Syntax:* 
```perl
$cmdr->pauseScheduler (<paused>);
```

*Example*

```perl
$cmdr->pauseScheduler (true);
```

**ectool**

*Syntax:* 
```bash
ectool pauseScheduler <paused>
```

*Example*

```bash
ectool pauseScheduler true
```
API Commands - Server Management

- **deleteLicense** on page 580
- **getAdminLicense** on page 581
- **getCertificates** on page 582
- **getLicense** on page 582
- **getLicenses** on page 583
- **getLicenseUsage** on page 584
- **getServerInfo** on page 584
- **getServerStatus** on page 585
- **getVersions** on page 586
- **getVersions** on page 586
- **logMessage** on page 588
- **setLogLevel** on page 588
- **shutdownServer** on page 589
- **tunePerformance** on page 590

**deleteLicense**

Deletes a license.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureName</td>
<td>The name of the licensed feature. Possible features include: Server</td>
</tr>
<tr>
<td>productName</td>
<td>The name of the product with the licensed feature. Possible products include: ElectricFlow</td>
</tr>
</tbody>
</table>

**Positional arguments**

productName, featureName

**Response**

None or a status OK message.

**ec-perl**

**syntax**: $cmdr->deleteLicense(<productName>, <featureName>);

**Example**

```perl
$cmdr->deleteLicense("ElectricFlow", "Server");
```
ECTOOL

**Syntax:**
```plaintext
ectool deleteLicense <productName> <featureName>
```

**Example**
```
ectool deleteLicense ElectricFlow Server
```

---

**getAdminLicense**

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

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

**Positional arguments**

None.

**Response**

You can receive one or more responses, depending on how you are licensed and actual license usage at the time of your query.

**Response examples:**

When the user does not have the necessary permission to use the Administrator license:

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

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

```
<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>
```
**getCertificates**

Returns the certificates in the trust chain for the server.

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

**Positional arguments**

None.

**Response**

None or a status OK message.

**ec-perl**

`syntax:`

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

**Example**

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

**ectool**

`syntax:`

```
ectool getAdminLicense
```

**Example**

```
ectool getAdminLicense
```

**getLicense**

Retrieves information for one license.

You must specify the `productName` and `featureName`. 
### ElectricFlow Perl API Commands

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>featureName</td>
<td>The name of the licensed feature. Possible features include: Server</td>
</tr>
<tr>
<td>productName</td>
<td>The name of the product with the licensed feature. Possible products include: ElectricFlow</td>
</tr>
</tbody>
</table>

**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

**getLicenses**

Retrieves all license data.

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

**Positional arguments**

None.

**Response**

Zero or more license elements.

**ec-perl**

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

*Example*

$cmdr->getLicenses();
getLicenseUsage

Retrieves the current license usage.

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

**Positional arguments**

None.

**Response**

You 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
```

getServerInfo

Returns information about server ports and message delivery.

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

**Positional arguments**

None.
**Response**

Returns the information about the server.

**ec-perl**

```perl
$scmdr->getServerInfo;
```

**Examples**

```
$scmdr->getServerInfo();
```

**ectool**

```shell
ectool getServerInfo
```

**Examples**

```
ectool getServerInfo
```

## getServerStatus

Retrieves the current status of the ElectricFlow server.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>
| diagnostics        | `<Boolean flag - 0|1|true|false>`<br>Select the diagnostic information that you want in your output:  
  - threadDump─stack dumps of all threads in the server  
  - statistics─output from all system timers  
  - systemProperties─values of all java system properties  
  - environmentVariables─values of all environment variables  
  - settings─values of all server settings  
  - serverInfo─output from getServerInfo call. |
| serverStateOnly    | `<Boolean flag - 0|1|true|false>`<br>If the argument is set to 1, 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. |
### Arguments and Descriptions

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>timeout</td>
<td>This argument specifies the timeout for the <code>element</code> flag. The default value is 120 seconds.</td>
</tr>
</tbody>
</table>

#### Positional Arguments

None.

#### Response

Returns the current status of the server, including the log message generated during the startup sequence.

This command returns different information depending on when and how it is called.

**Note:** You will get a lengthy response if you connect with a session that has admin privileges or if the server is still in a bootstrap state. After the server enters the "running" state, it is able to perform access checks but displays only the short form until you log in.

A simple response:

```
<serverState>running</serverState>
```

For more detailed server status response information, click [here](#).

#### ec-perl Syntax

**Syntax:**

```perl
$cmdr->getServerStatus({<optionals>});
```

**Examples**

- `$cmdr->getServerStatus();`
- `$cmdr->getServerStatus({diagnostics=>1});`

#### ectool Syntax

**Syntax:**

```bash
ectool getServerStatus
```

**Examples**

- `ectool getServerStatus`
- `ectool getServerStatus --diagnostics 1`

---

**getVersions**

Retrieves server version information.

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

Response
A serverVersion element.

ec-perl

.syntax: $cmdr->getVersions();

Example
$cmdr->getVersions();

ectool

.syntax: ectool getVersions

Example
ectool getVersions

importLicenseData
Imports one or more licenses.
You must specify licenseData.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>licenseData</td>
<td>The content of a license file (perl</td>
</tr>
<tr>
<td>licenseFile</td>
<td>&lt;localFileName&gt; The license file to import. This is a local file that will be read by ectool. The contents is sent as the licenseData argument (ectool only).</td>
</tr>
</tbody>
</table>

Positional arguments
licenseData

Response
None or a status OK message.

ec-perl

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

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

ectool

.syntax: ectool importLicenseData <licenseData>
**logMessage**

Enters a message in the server log.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>Message to add to the server log.</td>
</tr>
<tr>
<td>level</td>
<td>Select a message level: TRACE, DEBUG, INFO, WARN, ERROR, or OFF.</td>
</tr>
<tr>
<td>logger</td>
<td>Name of the object that logged the message.</td>
</tr>
</tbody>
</table>

**Positional arguments**

message

**Response**

None or a status OK message.

**ec-perl**

*syntax:* `$cmdr->logMessage (<message>, <optionals>));`

*Examples*

```perl
$cmdr->logMessage ("abort job step" {level => INFO});
```

**ectool**

*syntax:* `ectool logMessage <message> [optionals...]`

*Examples*

```bash
 ectool logMessage "abort job step" --level INFO
```

**setLogLevel**

Changes log level of a logger.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>logger</td>
<td>Name of the user or resource that logged the message.</td>
</tr>
<tr>
<td>level</td>
<td>Select a message level: TRACE, DEBUG, INFO, WARN, ERROR, or OFF.</td>
</tr>
</tbody>
</table>
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

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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>force</td>
<td>**&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>restart</td>
<td>**&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

Positional arguments
NoneResponse
None or a status OK message.

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

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

**ectool**

**Syntax:** ectool shutdownServer ...

**Example**

`ectool shutdownServer --restart 1`

**Back to Top**

---

## tunePerformance

Adjusts how the server is performing.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiQueueSize</td>
<td>Maximum number of threads that will be created in the API thread pool.</td>
</tr>
<tr>
<td>dbConnectionPoolSize</td>
<td>Maximum number of database connections that the server will use.</td>
</tr>
<tr>
<td>dbThreadPoolSize</td>
<td>Number of worker threads in the database connection manager.</td>
</tr>
<tr>
<td>dispatchQueueSize</td>
<td>Maximum number of threads that will be created in the Dispatch thread pool.</td>
</tr>
<tr>
<td>quartzQueueSize</td>
<td>Maximum number of threads that will be created in the Quartz thread pool.</td>
</tr>
<tr>
<td>stateMachineQueueSize</td>
<td>Maximum number of threads that will be created in the stateMachine thread pool.</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

None or a status OK message.

**ec-perl**

**Syntax:** $cmdr->tunePerformance {{optionals}};

**Examples**

```perl
$cmdr->tunePerformance ({apiQueueSize => 4, dbConnectionPoolSize => 2});
```

**ectool**

**Syntax:** ectool tunePerformance [optionals ...]
**Examples**

ectool tunePerformance --apiQueueSize 4 --dbConnectionPoolSize 2

**API Commands - Snapshots**

- `createSnapshot` on page 591
- `deleteSnapshot` on page 592
- `getPartialApplicationRevision` on page 593
- `getSnapshot` on page 594
- `getSnapshotEnvironments` on page 595
- `getSnapshots` on page 596
- `modifySnapshot` on page 597

**createSnapshot**

Creates a new snapshot of the specified application.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>The name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>snapshotName</code></td>
<td>The name of the snapshot that must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>componentVersions</code></td>
<td>(Optional) The name and version of the component for the snapshot.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Map</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>(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 &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code>. Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) Name of environment from which the snapshot is created. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, applicationName, snapshotName

**Response**

None or a status OK message.

**ec-perl**

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

**Example**

$cmdr->createSnapshot ("Build and Run", "Deploy", "Test Run" {description => "Beta only"});

**ectool**

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

**Example**

ectool createSnapshot "Build and Run" "Deploy" "Test Run" --description "Beta only"

**deleteSnapshot**

Deletes snapshot from an application.

You must specify projectName, applicationName, and snapshotName.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>The name of the snapshot that must be unique within the application. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, applicationName, snapshotName

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* $cmdr->deleteSnapshot (<projectName>, <applicationName>, <snapshotName>);

*Example*

$cmdr->deleteSnapshot ("Build and Run", "Undeploy", "Alpha");

**ectool**

*Syntax:* ectool deleteSnapshot <projectName> <applicationName> <snapshotName>

*Example*

ectool deleteSnapshot "Build and Run" "Undeploy" "Alpha"

**getPartialApplicationRevision**

Gets a partial application when a snapshot is created.

You must specify projectName, applicationName, and revisionNumber.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects. Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>revisionNumber</td>
<td>The revision number of the application.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
</tbody>
</table>

### Positional arguments

- `projectName`, `applicationName`, `revisionNumber`  

### Response

A list of environments deployed in the snapshot.

### ec-perl

**Syntax:**

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

**Example**

```
$cmdr->getPartialApplicationRevision ("Demo Project", "Deploy", 2);
```

### ectool

**Syntax:**

```
ectool getPartialApplicationRevision <projectName> <applicationName> <revisionNumber>
```

**Example**

```
ectool getPartialApplicationRevision "Demo Project" "Deploy" 2
```

### getSnapshot

Find a snapshot by name.

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>The application that owns the deployment history item.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>snapshotName</td>
<td>The name of the snapshot that must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

#### Positional arguments

- projectName, applicationName, snapshotName

#### Response

One snapshot element.

#### ec-perl

**Syntax:**

```perl
$cmdr->getSnapshot (<projectName>, <applicationName>, <snapshotName>);
```

**Example**

```perl
$cmdr->getSnapshot ("Demo Project", "Deploy", "Production");
```

#### ectool

**Syntax:**

```bash
ectool getSnapshot <projectName> <applicationName> <snapshotName>
```

**Example**

```bash
ectool getSnapshot "Demo Project" "Deploy" "Production"
```

### getSnapshotEnvironments

Gets a list of environments deployed in the specified snapshot.

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

#### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>The name of the snapshot that must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

#### 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 ("Demo Project", "Deploy", "Production Run");

ectool

**syntax:**
ectool getSnapshotEnvironments <projectName> <applicationName> <snapshotName>

**Example**
ectool getSnapshotEnvironments "Demo Project" "Deploy" "Production Run"

getSnapshots

Finds all the snapshots in an application.
You must specify projectName and applicationName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application that must be unique among all projects. Argument type: String</td>
</tr>
</tbody>
</table>

Positional arguments

projectName, applicationName

Response
Zero or more snapshot elements.

ec-perl

**syntax:**
$cmdr->getSnapshots (<projectName>, <applicationName>);

**Example**
$cmdr->getSnapshots ("Demo Project", "Deploy");

ectool

**syntax:**
ectool getSnapshots <projectName> <applicationName>
**modifySnapshot**

Creates a new snapshot of the specified application.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>The name of the application that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>snapshotName</code></td>
<td>The name of the snapshot that must be unique within the application.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>componentVersions</code></td>
<td>(Optional) Name and version of the component for the snapshot.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Map</td>
</tr>
<tr>
<td><code>description</code></td>
<td>(Optional) Comment text describing this object, which is not interpreted by</td>
</tr>
<tr>
<td></td>
<td>ElectricFlow. A plain text or HTML description for this object.</td>
</tr>
<tr>
<td></td>
<td>If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags.</td>
</tr>
<tr>
<td></td>
<td>The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;dt&gt; &lt;i&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;id&gt; &lt;img&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;td&gt; &lt;th&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;tr&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>environmentName</code></td>
<td>(Optional) Name of environment from which snapshot is created.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>newName</code></td>
<td>(Optional) New name of the snapshot.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `applicationName`
**Response**
None or a status OK message.

**ec-perl**

*Syntax:* $cmdr->createSnapshot (<projectName>, <applicationName>, {<optionals>});

*Example*

$cmdr->createSnapshot ("Build and Run", "Deploy", {description => "Internal use only", newName => "Trial Run"});

**ectool**

*Syntax:* ectool createSnapshot <projectName> <applicationName> [optionals ...]

*Example*

ectool createSnapshot "Build and Run" "Deploy" --description "Beta only" --newName "Trial Run"

---

**API Commands - Tier Map**

- [createTierMap](#) on page 598
- [deleteTierMap](#) on page 599
- [deleteTierMapping](#) on page 600
- [getTierMaps](#) on page 601
- [modifyTierMap](#) on page 602

---

### createTierMap

Creates a new tier map for an application.

**Required Arguments**

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

- `applicationName`
  
  **Description:** Name of the application; must be unique among all applications in the project.
  
  **Argument Type:** String

- `environmentProjectName`
  
  **Description:** Name of the environment's project; must be unique among all projects.
  
  **Argument Type:** String

- `environmentName`
  
  **Description:** Name of the environment; must be unique among all applications in the project.
Argument Type: String

Optional Arguments

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", "newApp", "defaultEnv", "Env1",
  {tierMapping => [{applicationTier => "AppTier1",
    environmentTier => "EnvTier1"}, {applicationTier => "AppTier2",
    environmentTier => "EnvTier2"}], tierMapName => "TierMap1"});

ectool

Syntax:

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

Example:

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

deleteTierMap

Deletes a tier map from an application.

Required Arguments

projectName
**deleteTierMapping**

Deletes a tier mapping from a tier map.

**Required Arguments**

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

- **applicationName**
  - **Description:** Name of the application; must be unique among all applications in the project.
  - **Argument Type:** String
Argument Type: String

environmentProjectName

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

environmentName

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

applicationTierName

Description: Name of the application tier.
Argument Type: String

Optional Arguments
None

Response
Deletes the specified tier mapping.

ec-perl
Syntax:
$<object>-deleteTierMapping(<projectName>, <applicationName>, <environmentProjectName>, <environmentName>, <applicationTierName>);

Example:
$sec->deleteTierMap("default", "Appl", "MyProj", "Env1", "InstallTier");

ectool
Syntax:
ectool deleteTierMapping <projectName> <applicationName> <environmentProjectName> <environmentName> <applicationTierName>

Example:
ectool deleteTierMapping default TierMapToDelete defaultEnv Env1 InstallTier

getTierMaps

Retrieves all tier maps that are used by the given application.

Required Arguments

projectName

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

applicationName

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

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 has the most recently used environment in the tier maps.
Argument Type: Boolean

Response

Returns a list of tier maps.

c perl

Syntax:

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

Example:

```perl
Sec->getTierMaps("default", "NewApp", {applicationEntityRevisionId => "4fa765dd-73f1-11e3-b67e-b0a420524153");
```

c tool

Syntax:

```bash
ectool getTierMaps <projectName> <applicationName> [optionals...]
```

Example:

```bash
ectool getTierMaps default NewApp --applicationEntityRevisionId 4fa765dd-73f1-11e3-b67e-b0a420524153
```

modifyTierMap

Modifies an existing tier map.

Required Arguments

projectName

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

applicationName

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

environmentProjectName

Description: Name of the environment's project; must be unique among all projects.
Argument Type: String
Description: Name of the environment; must be unique among all applications in the project.

Argument Type: String

Optional Arguments

tierMapName

Description: New name of the tier map, if specified.

Argument Type: String

tierMapping

Description: List of mappings between the application tiers and the environment tiers. The list shows the mappings as <applicationTier>=<environmentTier>.

If you use this argument, new tier mappings are added or existing mappings are updated for the specified application tiers. This argument does not replace all the mappings and thus does not remove the mappings that were not specified in the API call. To remove mappings, use the deleteTierMapping command.

Argument Type: Map

Response
Retrieves the updated tier map.

ec-perl

Syntax:

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

Example:

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

ectool

Syntax:

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

Example:

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

API Commands - User and Group Management

addUsersToGroup on page 604
createGroup on page 605
createUser on page 606
deleteGroup on page 607
deleteUser on page 607
getGroup on page 608
addUsersToGroup

Adds ones or more specified users to a particular group.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>groupName</code></td>
<td>The name of the group you are modifying.</td>
</tr>
<tr>
<td><code>userName</code></td>
<td>Using ec-perl, enter one or more user names to add to the group.</td>
</tr>
<tr>
<td><code>userName</code></td>
<td>Using ectool, enter one user name to add to the group.</td>
</tr>
<tr>
<td><code>userNames</code></td>
<td>Using ectool, enter two or more user names to add to the group.</td>
</tr>
</tbody>
</table>

**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**

```perl
$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 userName argument to add two or more users to the group.

ectool addUsersToGroup Developers --userNames John Jim Joey

createGroup

Creates a new local group of users.

You must specify a groupName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>groupName</td>
<td>Name for the new group that you are creating.</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) Using ec-perl, enter one or more user names to add to the group.</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) Using ectool, enter one user name to add to the group.</td>
</tr>
<tr>
<td>userNames</td>
<td>(Optional) Using ectool, enter two or more user names to add to the group.</td>
</tr>
</tbody>
</table>

**Positional arguments**

groupName

**Response**

None or status OK message.

**ec-perl**

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

**Example**

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

**ectool**

*syntax:* ectool createGroup <groupName> --userNames <user1> ...

**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"
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 *groupName*. The commands *createGroup* and *modifyGroup* can have an optional argument named *userNames*. In each case, the optional argument is followed by a list of groups or names.

Using ectool, your command string would be:

```
ectool createGroup "New Group Name" --userNames "A Adams" "B Barker"
```

To make this call via the Perl API, create a list of names and then pass a reference to the list as an optional parameter.

**Note:** The name of the optional parameter is singular, "*userName" or "*userGroup," not the plural form used by ectool.

Here is an example using the Perl API:

```perl
# Run the procedure - pass a reference to the list of names
$xPath = $cmdr->createGroup("New Group Name", {"userName" => ['A Adams', 'B Burns']});
```

You must specify a *userName*.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>The email address of the user.</td>
</tr>
<tr>
<td>fullUserName</td>
<td>Full name of the user, not a nickname.</td>
</tr>
<tr>
<td>groupNames</td>
<td>List of groups of which this user is a member. Any group name containing spaces must be enclosed in double-quotes.</td>
</tr>
<tr>
<td>password</td>
<td>The password of the user.</td>
</tr>
<tr>
<td>userName</td>
<td>This could be the user's full name, but more commonly it is the shortened name, first initial and last name, or nickname used for email.</td>
</tr>
</tbody>
</table>

**Positional arguments**

*userName*

**Response**

None or a status OK message.

**ec-perl**

```perl
syntax: $cmdr->createUser(<userName>, {<optionals>});
```
**Example**

```perl
$cmdr->createUser("aallen", {fullUserName => "Albert Allen");
```

**ectool**

*syntax:* `ectool createUser <userName> ...`

**Examples**

- `ectool createUser "aallen" --fullUserName "Albert Allen"
- `ectool createUser "Betty Barker"

---

**deleteGroup**

Deletes a local group.

You must specify a `groupName`.

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

**Positional arguments**

`groupName`

**Response**

None or a status OK message.

**ec-perl**

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

**Example**

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

**ectool**

*syntax:* `ectool deleteGroup <groupName>

**Example**

`ectool deleteGroup "Build Users"

---

**deleteUser**

Deletes a local user.

You must specify the `userName`. 
Arguments | Descriptions
---|---
userName | The name of the user you want to delete.

**Positional arguments**

userName

**Response**

None or a status OK message.

**ec-perl**

*Syntax:* `$cmdr->deleteUser(<userName>);`

*Example*

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

**ectool**

*Syntax:* `ectool deleteUser <userName>`

*Example*

```
ectool deleteUser "Betty Barker"
```

**getGroup**

Retrieves a group by its name.

You must specify the **groupName**.

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

**Positional arguments**

groupName

**Response**

One **group** element.

**ec-perl**

*Syntax:* `$cmdr->getGroup(<groupName>, {<optionals>});`
**Example**

```perl
$cmdr->getGroup("myGroup", {providerName => "LDAP"});
```

**ectool**

- **syntax:** `ectool getGroup <groupName> ...`

**Example**

```bash
ectool getGroup myGroup --providerName LDAP
```

## getGroups

Retrieves all groups.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>A string used to filter the returned groups by their names.</td>
</tr>
<tr>
<td>includeAll</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>maximum</td>
<td>Specifies the maximum number of groups you want to see.</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Zero or more `group` elements, each containing summary information only.

**ec-perl**

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

**Example**

```perl
$cmdr->getGroups({filter => "dev*", maximum => 3,});
```

**ectool**

- **syntax:** `ectool getGroups ...`

**Example**

```bash
ectool getGroups --filter dev* --maximum 3
```
getUser

Retrieves a user by name.
You must specify the userName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerName</td>
<td>The name of the directory provider. If specified, this option limits the search to the specified directory provider.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**userName**

**Response**

One user element.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* ectool getUser <userName> ...

*Example*

ectool getUser "Betty Barker"

getUsers

Retrieves 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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>&lt;filter pattern&gt; Enter a filter pattern to match user names. The filter is not case sensitive and can include the &quot;*&quot; wildcard character. Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments and Descriptions

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeAll</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>maximum</td>
<td>&lt;number of users&gt; Specify a larger number of user records to retrieve. The default limit is 100 user records. Argument type: Integer</td>
</tr>
<tr>
<td>searchFields</td>
<td>Filter search fields that include the user's full name and email address. Argument type: Collection</td>
</tr>
</tbody>
</table>

### Positional arguments

None

### Response

Zero or more user elements with summary information only.

### ec-perl

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

**Examples**

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

$cmdr->getUsers({filter => '*Betty*', maximum => 25});
```

### ectool

**syntax:** ectool getUsers ...

**Examples**

```bash
ectool getUsers

ectool getUsers --filter *Betty* --maximum 25
```

### login

Logs into the server and saves the session ID for subsequent ectool use. The user name provided determines the permissions for commands that can be run during the session.

You must specify the **userName** and **password**.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>password</td>
<td>The password for the user who is &quot;logging in&quot;.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of a user who has login privileges.</td>
</tr>
</tbody>
</table>

**Positional arguments**

userName, password

**Response**

One session element containing the session ID.

**ec-perl**

```perl
syntax: $cmdr->login(<userName>,<password>);
```

*Example*

```
$cmdr->login("Ellen Ernst", "ee123")
```

**ectool**

```bash
syntax: ectool login <userName> <password>
```

*Note:* ectool will prompt for the password if not supplied.

*Example*

```
ectool --server EAVMXP login "Ellen Ernst" "ee123"
```

**logout**

Logs out of the client session.

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

**Positional arguments**

None

**Response**

None or a status OK message.

**ec-perl**

*Example*

```
$cmdr->logout();
```
**modifyGroup**

Modifies an existing group.

You must specify `groupName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>groupName</code></td>
<td>Name of the group that must be unique among local groups.</td>
</tr>
<tr>
<td><code>migrateSettings</code></td>
<td>(Optional) <code>&lt;targetGroupName&gt;</code> Use this argument to specify the new name to which the settings need to be moved.</td>
</tr>
<tr>
<td><code>newName</code></td>
<td>(Optional) Enter any name of your choice to rename the group.</td>
</tr>
<tr>
<td><code>removeAllUsers</code></td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><code>userName</code></td>
<td>(Optional) Using ec-perl, enter one or more user names to add to the group.</td>
</tr>
<tr>
<td></td>
<td><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.</td>
</tr>
<tr>
<td><code>userName</code></td>
<td>(Optional) Using ectool, enter one user name to add to the group.</td>
</tr>
<tr>
<td></td>
<td><code>user1</code> 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.</td>
</tr>
<tr>
<td><code>userNames</code></td>
<td>(Optional) Using ectool, enter two or more user names to add to the group.</td>
</tr>
<tr>
<td></td>
<td><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.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`groupName`

**Response**

None or a status OK message.
**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:

```
ecctool 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`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>The user's email address.</td>
</tr>
<tr>
<td>fullUserName</td>
<td>The user's full name. For example, &quot;John Smith&quot;.</td>
</tr>
</tbody>
</table>
### Arguments | Descriptions
--- | ---

<table>
<thead>
<tr>
<th><strong>Arguments</strong></th>
<th><strong>Descriptions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>groupNames</td>
<td><em>group1</em> [<em>group2</em> ...] Assigns the user to one or more groups and removes the user from any groups not included in the list.</td>
</tr>
<tr>
<td>migrateSettings</td>
<td><code>&lt;targetUserName&gt;</code> Use this option to specify the new name to which the settings need to be moved.</td>
</tr>
<tr>
<td>newName</td>
<td>The user's new name (for example, if changing an existing user's surname).</td>
</tr>
<tr>
<td>password</td>
<td>Enter a new password to set for the user.</td>
</tr>
<tr>
<td>removeFromAllGroups</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>sessionPassword</td>
<td>If changing the user's password, you must enter the password used in the &quot;login&quot; command also.</td>
</tr>
<tr>
<td>userName</td>
<td>The name used by the user to login and/or receive email. For example, &quot;jsmith&quot;.</td>
</tr>
</tbody>
</table>

### Positional arguments

**userName**

### Response

None or a status OK message.

### ec-perl

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

**Example**

```
$cmdr->modifyUser("Betty Barker", {email => "bbarker@abc.com"});
```

### ectool

**syntax:** `ectool modifyUser <userName> ...

**Example**

```
 ectool modifyUser "Betty Barker" --email "bbarker@abc.com"
```

### removeUsersFromGroup

Removes one or more users from a particular group.

You must specify a `groupName` and one or more user names.
Arguments | Descriptions
---|---
groupName | The name of the group from which to remove users.
userNames | The list of users to remove from the group.

**Positional arguments**
groupName, userNames

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

**ec-perl**

* syntax:*

```
$cldr->removeUsersFromGroup(<groupName>, {<optionals>});
```

* Example *

```
$cldr->removeUsersFromGroup("Developers", {userName => ["John", "Jim", "Joey"]});
```

**ectool**

* syntax:*

```
ectool removeUsersFromGroup <groupName> <userNames> ...
```

* Example *

```
ectool removeUsersFromGroup Developers --userNames John Jim Joey
```

---

**API Commands - Workflow Management**

- `completeWorkflow` on page 616
- `deleteWorkflow` on page 617
- `getState` on page 618
- `getStates` on page 619
- `getTransition` on page 619
- `getTransitions` on page 620
- `getWorkflow` on page 621
- `getWorkflows` on page 622
- `runWorkflow` on page 622
- `transitionWorkflow` on page 623

**completeWorkflow**

Marks a workflow as completed. When completed, transitions are no longer evaluated.

You must specify `projectName` and `workflowName`. 
### completeWorkflow

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `projectName`, `workflowName`

**Response**

None or status OK message.

**ec-perl**

**syntax:**

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

**Example**

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

**ectool**

**syntax:**

```
ectool completeWorkflow <projectName> <workflowName>
```

**Example**

```
ectool completeWorkflow projectA workflow_26_201010121647
```

### deleteWorkflow

Deletes a workflow, including all states and transitions.

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

**Arguments**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteProcesses</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow to delete.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- `projectName`, `workflowName`
Response
None or status OK message.

ec-perl

Syntax: $cmdr->deleteWorkflow (<projectName>, <workflowName>);

Example
$cmdr->deleteWorkflow ("projectA", "workflow_26_201010121647");

ectool

Syntax: ectool deleteWorkflow <projectName> <workflowName> ...

Example
ectool deleteWorkflow projectA workflow_26_201010121647

getState

Finds a state by name.

You must specify projectName, workflowName, and stateName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>stateName</td>
<td>Name of the state.</td>
</tr>
<tr>
<td>workflowName</td>
<td>Name of the workflow.</td>
</tr>
</tbody>
</table>

Positional arguments
projectName, workflowName, stateName

Response
One state element.

ec-perl

Syntax: $cmdr->getState (<projectName>, <workflowName>, <stateName>);

Example
$cmdr->getState ("projectA", "workflow_26_201010121647", "build");

ectool

Syntax: ectool getState <projectName> <workflowName> <stateName>

Example
ectool getState projectA workflow_26_201010121647 build
getStates

Retrieves all states in a workflow.

You must specify `projectName` and `workflowName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>workflowName</td>
<td>Name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `workflowName`

**Response**

One or more `state` elements.

**ec-perl**

*Syntax:* `$cmdr->getStates (projectName>, <workflowName>);

*Example*

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

**ectool**

*Syntax:* `ectool getStates <projectName> <workflowName>

*Example*

```bash
ectool getStates projectA workflow_26_201010121647
```

getTransition

Finds a transition by name.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>stateName</td>
<td>Name of the state.</td>
</tr>
<tr>
<td>transitionName</td>
<td>Name of the transition.</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---
workflowName | Name of the workflow.

**Positional arguments**

`projectName, workflowName, stateName, transitionName`

**Response**

One `transition` element.

**ec-perl**

*Syntax:* `$cmdr->getTransition (projectName>, <workflowName>, <stateName>, <transitionName>);`

*Example*

```
$cmdr->getTransition ("projectA", "workflow_26_201010121647", "build", "build2test");
```

**ectool**

*Syntax:* `ectool getTransition <projectName> <workflowName> <stateName> <transitionName>`

*Example*

```
ectool getTransition projectA workflow_26_201010121647 build build2test
```

**Back to Top**

getTransitions

Retrieves all transitions in a workflow.

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

Arguments | Descriptions
--- | ---
projectName | The name of the project containing the transition.
stateName | The name of the state.
targetState | The target state for the transition definition.
workflowName | The name of the workflow.

**Positional arguments**

`projectName, workflowName, stateName`
Response
One or more transition elements.

ec-perl

**syntax:** $cmdr-\>getTransitions (<projectName>, <workflowName>, <stateName>);

**Example**
$cmdr-\>getTransitions ("projectA", "workflow_26_201010121647", "build");

ectool

**syntax:** ectool getTransitions <projectName> <workflowName> <stateName>

**Example**
ectool getTransitions projectA workflow_26_201010121647 build

getWorkflow

Finds a workflow by name.
You must specify a **projectName** and **workflowName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>workflowName</td>
<td>Name of the workflow.</td>
</tr>
</tbody>
</table>

**Positional arguments**
projectName, workflowName

Response
One workflow element.

ec-perl

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

**Example**
$cmdr-\>getWorkflow ("projectA", "BTD");

ectool

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

**Example**
ectool getWorkflow projectA BTD
getWorkflows

Retrieves all workflow instances in a project.

You must specify a projectName.

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

Positional arguments

projectName

Response

Zero or more workflow elements.

ec-perl

* syntax: $cmdr-&gt;getWorkflows (&lt;projectName&gt;);

* Example

  $cmdr-&gt;getWorkflows ("projectA");

ectool

* syntax: ectool getWorkflows &lt;projectName&gt;

* Example

  ectool getWorkflows projectA

runWorkflow

Runs the specified workflow definition and returns the workflow name.

You must specify the projectName and workflowDefinitionName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>Name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>actualParameter</td>
<td>(Optional) Specifies the values to pass as parameters to the workflow starting state. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the starting state. Argument type: Map</td>
</tr>
<tr>
<td>credentials</td>
<td>(Optional) Credentials to use with the workflow state. Argument type: Collection</td>
</tr>
<tr>
<td>priority</td>
<td>(Optional) Priority of the jobs launched by the workflow. Argument type: JobPriority</td>
</tr>
<tr>
<td>startingState</td>
<td>(Optional) The initial state of the workflow. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, workflowDefinitionName

**Response**

The workflow name is returned.

**ec-perl**

*Syntax:* $cmdr->runWorkflow (<projectName>, <workflowDefinitionName>, [<optionals>]);

*Example*

$cmdr->runWorkflow ("projectA", "BTD", {startingState => "build"});

**ectool**

*Syntax:* ectool runWorkflow <projectName> <workflowDefinitionName> ...

*Example*

ectool runWorkflow projectA BTD --startingState build

**transitionWorkflow**

Manually transition from the active workflow state.

You must specify projectName, workflowName, stateName, and transitionName.
### Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow to transition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition. Argument type: String</td>
</tr>
<tr>
<td>actualParameters</td>
<td>(Optional) Specifies the values to pass as parameters to the transition's target state. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the target state. Argument type: Map</td>
</tr>
<tr>
<td>credentials</td>
<td>(Optional) Credentials to use with the workflow state. Argument type: Collection</td>
</tr>
</tbody>
</table>

### Positional arguments

- `projectName`, `workflowName`, `stateName`, `transitionName`

### Response

None or status OK message.

### ec-perl

**Syntax:**

```perl
$cmdr->transitionWorkflow (<projectName>, <workflowName>, <stateName>, <transitionName>, [{optionals}]);
```

**Example**

```perl
$cmdr->transitionWorkflow ("projectA", "workflow_26_201010121647", "build", "build2 test");
```

### ectool

**Syntax:**

```bash
ectool transitionWorkflow <projectName> <workflowName> <stateName> <transitionName> ...
```

**Example**

```bash
ectool transitionWorkflow projectA workflow_26_201010121647 build build2test
```
API Commands - Workflow Definition Management

`createStateDefinition` on page 625
`createTransitionDefinition` on page 627
`createWorkflowDefinition` on page 629
`deleteStateDefinition` on page 630
`deleteTransitionDefinition` on page 630
`deleteWorkflowDefinition` on page 631
`getStateDefinition` on page 632
`getStateDefinitions` on page 633
`getTransitionDefinition` on page 633
`getTransitionDefinitions` on page 634
`getWorkflowDefinition` on page 635
`getWorkflowDefinitions` on page 636
`modifyStateDefinition` on page 636
`modifyTransitionDefinition` on page 638
`modifyWorkflowDefinition` on page 639
`moveStateDefinition` on page 640
`moveTransitionDefinition` on page 641

`createStateDefinition`

Creates a new state definition for a workflow definition. Optionally, a state may launch either a procedure or a sub-workflow as its "process" when the state is entered.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>The name of the project.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>workflowDefinitionName</code></td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>stateDefinitionName</code></td>
<td>Choose any unique name of your choice for the state definition.</td>
</tr>
<tr>
<td></td>
<td>This name must be unique within the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
## Arguments

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>(Optional) Specifies the values to pass as parameters to the process. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the process. For more information about parameters, click here. Argument type: Map</td>
</tr>
<tr>
<td>description</td>
<td>(Optional) A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code>. Argument type: String</td>
</tr>
<tr>
<td>startable</td>
<td>(Optional) `&lt;Boolean flag = 0</td>
</tr>
<tr>
<td>subprocedure</td>
<td>(Optional) Name of the procedure launched when the state is entered. <strong>Also requires</strong> subproject. Argument type: String</td>
</tr>
<tr>
<td>subproject</td>
<td>(Optional) Name of the project containing the procedure or workflow launched when the state is entered. Argument type: String</td>
</tr>
<tr>
<td>substartingState</td>
<td>(Optional) Name of the starting state for the workflow launched when the state is entered. <strong>Also requires</strong> subproject and subworkflowDefinition. Argument type: String</td>
</tr>
<tr>
<td>subworkflowDefinition</td>
<td>(Optional) Name of the workflow definition launched when the state is entered. <strong>Also requires</strong> subproject. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, workflowDefinitionName, stateDefinitionName

**Response**

One stateDefinition element.
**ec-perl**

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

**Example**

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

**ectool**

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

**Example**

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

**createTransitionDefinition**

Creates a new transition definition for workflow definition.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>workflowDefinitionName</code></td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>stateDefinitionName</code></td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>transitionDefinitionName</code></td>
<td>The name of the transition that must be unique among all state definitions.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><code>targetState</code></td>
<td>Target state for the transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
---|---
actualParameter | (Optional) Specifies the values to pass as parameters to the target state. Each parameter value is specified with an `actualParameterName` and a value. The `actualParameterName` must match the name of a formal parameter on the target state. For more information about parameters, click here.
Argument type: Map

condition | (Optional) A fixed text or text embedding property references that are evaluated into a logical TRUE or FALSE. An empty string, a "0" or "false" is interpreted as FALSE. Any other result string is interpreted as TRUE. This field is ignored by the server if `trigger` is set to manual.
Argument type: String

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

trigger | (Optional) Type of trigger for this transaction.
Possible values are: `onEnter`, `onStart`, `onCompletion`, `manual`
Argument type: TransitionTrigger

Positional arguments
- `projectName`, `workflowDefinitionName`, `stateDefinitionName`, `transitionDefinitionName`, `targetState`

Response
One `transitionDefinition` element.

ec-perl

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

*Example*

```perl
$cmdr->createTransitionDefinition ("ProjectA", "BTD", "build", "build2test", "test",
    {trigger => "manual", description => "free text"});
```

ectool

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

Creates a new workflow definition for a project.
You must enter a `projectName` and a `workflowDefinitionName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt;</code> ... <code>&lt;html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tt&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>Name of the workflow definition; must be unique within the project.</td>
</tr>
<tr>
<td>workflowNameTemplate</td>
<td>The name of the workflow template that the system uses to name instances of this workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**
`projectName`, `workflowDefinitionName`

**Response**
One `workflowDefinition` element.

**ec-perl**

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

**Example**
```
$cmdr->createWorkflowDefinition ("projectA", "BTD", {description => "free text")};
```

**ectool**

**syntax:**
```
ectool createWorkflowDefinition <projectName> <workflowDefinitionName>
    [<optionals>]
```

**Example**
```
ectool createWorkflowDefinition projectA BTD --description "free text"
```
**deleteStateDefinition**

Deletes a state definition.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the state definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

**projectName**, **workflowDefinitionName**, **stateDefinitionName**

**Response**

None or status OK message.

**ec-perl**

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

**Example**

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

**ectool**

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

**Example**

```bash
tool deleteStateDefinition projectA BTD build
```

Back to Top

---

**deleteTransitionDefinition**

Deletes a transition definition.

You must specify a **projectName**, **workflowDefinitionName**, **stateDefinitionName**, **transitionDefinitionName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the transition definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
</tbody>
</table>
transitionDefinitionName | The name of the transition definition.
workflowDefinitionName | The name of the workflow definition.

Positional arguments
projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName

Response
None or status OK message.

c-p-rl
syntax: $cmdr->deleteTransitionDefinition ($projectName, $workflowDefinitionName, $stateDefinitionName, $transitionDefinitionName);

d Example
$cmdr->deleteTransitionDefinition ("projectA", "BTD", "build", "build2test");

ectool
syntax: ectool deleteTransitionDefinition $projectName $workflowDefinitionName $stateDefinitionName $transitionDefinitionName

d Example
ectool deleteTransitionDefinition projectA BTD build build2test

deleteWorkflowDefinition

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

You must specify a *projectName* and a *workflowDefinitionName*

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>The name of the project containing the workflow definition to delete.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

Positional arguments
projectName, workflowDefinitionName

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

*Syntax*: `$cmdr->deleteWorkflowDefinition (<projectName>, <workflowDefinitionName>);`

*Example*

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

**ectool**

*Syntax*: `ectool deleteWorkflowDefinition <projectName> <workflowDefinitionName>`

*Example*

```
ectool deleteWorkflowDefinition projectA BTD
```

---

**getStateDefinition**

Finds a state definition by name.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td><code>stateDefinitionName</code></td>
<td>Name of the state definition.</td>
</tr>
<tr>
<td><code>workflowDefinitionName</code></td>
<td>Name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, workflowDefinitionName, stateDefinitionName`

**Response**

One `stateDefinition` element.

**ec-perl**

*Syntax*: `$cmdr->getStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>);`

*Example*

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

**ectool**

*Syntax*: `ectool getStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName>`

*Example*

```
ectool getStateDefinition projectA BTD build
```
**getStateDefinitions**

Retrieves all state definitions in a workflow definition.
You must specify **projectName** and **workflowDefinitionName**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>includeFormalParameters</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>startableOnly</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

projectName, workflowDefinitionName

**Response**

One or more stateDefinition elements.

**ec-perl**

*Syntax:* $cmdr->getStateDefinitions (<projectName>, <workflowDefinitionName>, {<optionals>});

*Example*

$cmdr->getStateDefinitions ("projectA", "BTD", {startableOnly => 1});

**ectool**

*Syntax:* ectool getStateDefinitions <projectName> <workflowDefinitionName> ...

*Example*

ectool getStateDefinitions projectA BTD --startableOnly 1

**getTransitionDefinition**

Finds a transition definition by name.
You must specify **projectName**, **workflowDefinitionName**, **stateDefinitionName**, **transitionDefinitionName**.

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateDefinitionName</td>
<td>Name of the state definition.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>Name of the transition definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>Name of the workflow definition.</td>
</tr>
</tbody>
</table>

### Positional arguments
- projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName

### Response
One `transitionDefinition` element.

**ec-perl**

Syntax: `$cmdr->getTransitionDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, <transitionDefinitionName>);

**Example**
```
$cmdr->getTransitionDefinition ("projectA", "BTD", "build", "build2test");
```

**ectool**

Syntax: `ectool getTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName>`

**Example**
```
ectool getTransitionDefinition projectA BTD build build2test
```

### getTransitionDefinitions

Retrieves all transition definitions in a workflow definition.

You must specify `projectName, stateDefinitionName, workflowDefinitionName`.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>targetState</td>
<td>The name of the target state.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

### Positional arguments
- projectName, stateDefinitionName, workflowDefinitionName
**Response**
Zero or more `transitionDefinition` elements.

**ec-perl**

*Syntax:* `$cmdr->getTransitionDefinitions (<projectName>, <stateDefinitionName>, <workflowDefinitionName>, {<optionals>});`

*Example*

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

**ectool**

*Syntax:* `ectool getTransitionDefinitions <projectName> <stateDefinitionName> <workflowDefinitionName> ...`

*Example*

```
ectool getTransitionDefinitions projectA build BTD
```

**getWorkflowDefinition**

Finds a workflow definition by name.

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

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td><code>workflowDefinitionName</code></td>
<td>Name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `workflowDefinitionName`

**Response**
One `workflowDefinition` element.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* `ectool getWorkflowDefinition <projectName> <workflowDefinitionName>`

*Example*

```
ectool getWorkflowDefinition projectA BTD
```
**getWorkflowDefinitions**

Retrieves all workflow definitions in a project.

You must specify a `projectName`.

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

**Positional arguments**

`projectName`

**Response**

Zero or more `workflowDefinition` elements.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* `ectool getWorkflowDefinitions <projectName>`

*Example*

```
ectool getWorkflowDefinitions projectA
```

**modifyStateDefinition**

Modifies an existing state definition.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the process. Each parameter value is specified with an <code>actualParameterName</code> and a value. The <code>actualParameterName</code> must match the name of a formal parameter on the called process.</td>
</tr>
</tbody>
</table>
# ElectricFlow Perl API Commands

## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clearActualParameters</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</code></td>
</tr>
<tr>
<td>newName</td>
<td>The new name of your choice for the state definition.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>startable</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition to modify.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>The name of the procedure launched when the state is entered. &lt;br&gt;<code>Also requires</code> subproject</td>
</tr>
<tr>
<td>subproject</td>
<td>The name of the project containing the procedure or workflow launched when the state is entered.</td>
</tr>
<tr>
<td>substartingState</td>
<td>The name of the workflow starting state that is launched when the state is entered. &lt;br&gt;<code>Also requires</code> subproject and subworkflowDefinition</td>
</tr>
<tr>
<td>subworkflowDefinition</td>
<td>The name of the workflow definition launched when the state is entered. &lt;br&gt;<code>Also requires</code> subproject</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

### Positional arguments

- projectName, workflowDefinitionName, stateDefinitionName

### Response

- One `stateDefinition` element.

### ec-perl

**syntax:** $cmdr-&gt;modifyStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>);
Example

```php
$cmdr->modifyStateDefinition ("projectA", "BTD", "build",
    [startable => 1,
    subproject => "factory",
    subprocedure => "Master",
    description => "sample text"));
```

ectool

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

Example

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

modifyTransitionDefinition

Modifies an existing transition definition.

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

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameter</td>
<td>Specifies the values to pass as parameters to the target state. Each parameter value is specified with an actualParameterName and a value. The actualParameterName must match the name of a formal parameter on the target state.</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>condition</td>
<td>A fixed text or text embedded property references that are evaluated into a logical &quot;true&quot; or &quot;false&quot;. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as &quot;false&quot;. Any other result string is interpreted as &quot;true&quot;. This field is ignored by the server if trigger is set to manual.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
</tbody>
</table>
### Arguments and Descriptions

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>newName</td>
<td>A new name of your choice for the transition definition—must be a unique name within the workflow.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>targetState</td>
<td>The target state for the transition definition.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition to modify.</td>
</tr>
<tr>
<td>trigger</td>
<td>Possible values are: onEnter</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

### Positional arguments

projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName

### Response

One transitionDefinition element.

### ec-perl

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

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

### ectool

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

**Example**
```
ectool modifyTransitionDefinition projectA BTD build build2test
   --targetState deploy
   --trigger onCompletion
   --description "bypass all tests"
```

### modifyWorkflowDefinition

Modifies an existing workflow definition.
You must specify `projectName` and `workflowDefinitionName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;style&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code>.</td>
</tr>
<tr>
<td>newName</td>
<td>The new name of your choice for the workflow definition—must be a unique name within the workflow.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project; must be unique among all projects.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition to modify.</td>
</tr>
<tr>
<td>workflowNameTemplate</td>
<td>The template used to determine default names for workflows launched from a workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName, workflowDefinitionName`

**Response**

One `workflowDefinition` element.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* `ectool modifyWorkflowDefinition <projectName> <workflowDefinitionName> ...`

*Example*

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

Back to Top

**moveStateDefinition**

Moves a state definition within a workflow definition.
You must specify `projectName`, `workflowDefinitionName`, and `stateDefinitionName`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>beforeStateDefinition</td>
<td>Use this option to reorder state definitions in a workflow definition. The state definition (stateDefinitionName) will be moved to a position just before the state definition &quot;named&quot; by this option. If omitted, the state definition is moved to the end of the workflow definition.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the state definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition to move.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

`projectName`, `workflowDefinitionName`, `stateDefinitionName`

**Response**

None or status OK message.

**ec-perl**

**syntax:**

```perl
$cmdr->moveStateDefinition (<projectName>, <workflowDefinitionName>, <stateDefinitionName>, {<optionals>});
```

**Example**

```perl
$cmdr->moveStateDefinition ("projectA", "BTD", "deploy", 
{beforeStateDefinition => "test"});
```

**ectool**

**syntax:**

```
ectool moveStateDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> ...
```

**Example**

```
ectool moveStateDefinition projectA BTD deploy --beforeStateDefinition test
```

Back to Top

**moveTransitionDefinition**

Moves a transition definition within a workflow definition.

You must specify `projectName`, `workflowDefinitionName`, `stateDefinitionName`, and `transitionDefinitionName`.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>beforeTransitionDefinition</td>
<td>Use this option to move a transition definition in a workflow definition. The transition definition is moved to a position just before the transition definition named by this option. If omitted, the transition definition is moved to the end of the workflow definition.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project containing the transition definition.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition to move.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Positional arguments**

- projectName, workflowDefinitionName, stateDefinitionName, transitionDefinitionName

**Response**

None or status OK message.

**ec-perl**

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

*Example*

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

**ectool**

*Syntax:* ectool moveTransitionDefinition <projectName> <workflowDefinitionName> <stateDefinitionName> <transitionDefinitionName> ...

*Example*

```
ectool moveTransitionDefinition projectA BTD Build in--beforeTransitionDefinition out
```

**Back to Top**

**API Commands - Workspace Management**

- [createWorkspace](#) on page 643
- [deleteWorkspace](#) on page 645
- [getWorkspace](#) on page 645
- [getWorkspaces](#) on page 646
modifyWorkspace on page 646
resolveFile on page 648

createWorkspace

Creates a new workspace.

A workspace definition consists of three paths to access the workspace in various ways:

agentDrivePath

agentUncPath - The agent uses agentUncPath and agentDrivePath to compute the drive mapping needed to make agentDrivePath valid in the step (see examples below).

agentUnixPath

Examples for agentDrivePath and agentUncPath:

<table>
<thead>
<tr>
<th>agentDrivePath</th>
<th>agentUncPath</th>
<th>Result from running a step in &quot;job123&quot; that uses this workspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:\</td>
<td>\server\share</td>
<td>The agent maps \server\share to drive n: and runs the step in n:\job123.</td>
</tr>
<tr>
<td>N:\sub1</td>
<td>\server\share\dir1\sub1</td>
<td>The agent maps \server\share\dir1 to drive n:\sub1 and runs the step in n:\sub1\job123.</td>
</tr>
<tr>
<td>N:\sub1</td>
<td>\server\share\dir1</td>
<td>Invalid! No mapping can be deduced from this pair of values.</td>
</tr>
<tr>
<td>C:\ws</td>
<td>C:\ws</td>
<td>A local workspace on the agent. No drive mapping is needed. The job step runs in c:\ws\job123.</td>
</tr>
<tr>
<td>C:\ws</td>
<td></td>
<td>Same as if agentUncPath were set identical to agentDrivePath.</td>
</tr>
</tbody>
</table>

You must specify a workspaceName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>agentDrivePath</td>
<td>Drive-letter-based path used by Windows agents to access the workspace in steps.</td>
</tr>
<tr>
<td>agentUncPath</td>
<td>UNC path used by Windows 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.</td>
</tr>
<tr>
<td>agentUnixPath</td>
<td>UNIX path used by UNIX agents and Linux ElectricFlow web servers to access the workspace.</td>
</tr>
</tbody>
</table>
## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>Credential to use when connecting to a network location. credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object absolute (for example, &quot;/projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object’s project.</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;li&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>local</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workspaceDisabled</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The workspace name.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone where this workspace resides.</td>
</tr>
</tbody>
</table>

### Positional arguments

workspaceName

### Response

None or a status OK message.

### ec-perl

**syntax:**

```perl
$cmdr->createWorkspace(<workspaceName>, {<optionals>});
```

**Example**

```perl
$cmdr->createWorkspace('test', {agentDrivePath => 'c:/workspace',
    agentUncPath => 'c:/workspace',
    agentUnixPath => '/mnt/server/workspace'});
```

### ectool

**syntax:**

```
ectool createWorkspace <workspaceName> ...
```

**Example**

```bash
ectool createWorkspace test --agentDrivePath c:/workspace --agentUncPath c:/workspace --agentUnixPath '/mnt/server/workspace'
```
deleteWorkspace

Deletes a workspace.
You must specify the workspaceName.

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

Positional arguments

workspaceName

Response

None or a status OK message.

ec-perl

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

Example

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

ectool

syntax: ectool deleteWorkspace <workspaceName>

Example

ectool deleteWorkspace test

getWorkspace

Retrieves a workspace by name.
You must specify the workspaceName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspaceName</td>
<td>The name of the workspace to retrieve.</td>
</tr>
</tbody>
</table>

Positional arguments

workspaceName

Response

One workspace element.

ec-perl

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

ectool
  syntax: ectool getWorkspace <workspaceName>

Example
  ectool getWorkspace test

getWorkspaces
Retrieves all workspaces.

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

Positional arguments
  None

Response
  Zero or more workspace elements.

ec-perl
  syntax: $cmdr->getWorkspaces();

Example
  $cmdr->getWorkspaces();

ectool
  syntax: ectool getWorkspaces

Example
  ectool getWorkspaces

modifyWorkspace
Modifies an existing workspace.

A workspace definition consists of three paths to access the workspace in various ways:
  agentDrivePath
  agentUncPath - The agent uses agentUncPath and agentDrivePath to compute the drive mapping needed to make agentDrivePath valid in the step (see examples below).
agentUnixPath

Examples for agentDrivePath and agentUncPath:

<table>
<thead>
<tr>
<th>agentDrivePath</th>
<th>agentUncPath</th>
<th>Result from running a step in &quot;job123&quot; that uses this workspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>N:\</td>
<td>\server\share</td>
<td>The agent maps \server\share to drive n: and runs the step in n:\job123.</td>
</tr>
<tr>
<td>N:\sub1</td>
<td>\server\share\dir1\sub1</td>
<td>The agent maps \server\share\dir1 to drive n: and runs the step in n:\sub1\job123.</td>
</tr>
<tr>
<td>N:\sub1</td>
<td>\server\share\dir1</td>
<td>Invalid! No mapping can be deduced from this pair of values.</td>
</tr>
<tr>
<td>C:\ws</td>
<td>C:\ws</td>
<td>A local workspace on the agent. No drive mapping is needed. The job step runs in c:\ws\job123.</td>
</tr>
<tr>
<td>C:\ws</td>
<td></td>
<td>Same as if agentUncPath were set identical to agentDrivePath.</td>
</tr>
</tbody>
</table>

You must specify a workspaceName.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>agentDrivePath</td>
<td>Drive-letter-based path used by Windows agents to access the workspace in steps.</td>
</tr>
<tr>
<td>agentUncPath</td>
<td>UNC path used by Windows 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.</td>
</tr>
<tr>
<td>agentUnixPath</td>
<td>UNIX path used by UNIX agents and Linux ElectricFlow web servers to access the workspace.</td>
</tr>
<tr>
<td>credentialName</td>
<td>credentialName can be one of two forms: relative (for example, &quot;cred1&quot;) - the credential is assumed to be in the project that contains the request target object. absolute (for example, &quot;projects/BuildProject/credentials/cred1&quot;) - the credential can be from any specified project, regardless of the target object's project.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with <code>&lt;html&gt; ... &lt;/html&gt;</code> tags. The only HTML tags allowed in the text are: <code>&lt;a&gt;</code>, <code>&lt;b&gt;</code>, <code>&lt;br&gt;</code>, <code>&lt;div&gt;</code>, <code>&lt;dl&gt;</code>, <code>&lt;font&gt;</code>, <code>&lt;i&gt;</code>, <code>&lt;li&gt;</code>, <code>&lt;ol&gt;</code>, <code>&lt;p&gt;</code>, <code>&lt;pre&gt;</code>, <code>&lt;span&gt;</code>, <code>&lt;table&gt;</code>, <code>&lt;tc&gt;</code>, <code>&lt;td&gt;</code>, <code>&lt;th&gt;</code>, <code>&lt;tr&gt;</code>, <code>&lt;ul&gt;</code></td>
</tr>
<tr>
<td>local</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>newName</td>
<td>Enter any name of your choice to rename the workspace.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace to modify.</td>
</tr>
<tr>
<td>workspaceDisabled</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone where this workspace resides.</td>
</tr>
</tbody>
</table>

**Positional arguments**

workspaceName

**Response**

None or a status OK message.

**ec-perl**

syntax: `$cmdr->modifyWorkspace(<workspaceName>, {<optionals>});`

*Example*

$cmdr->modifyWorkspace("test", {description => "my test workspace");

**ectool**

syntax: `ectool modifyWorkspace <workspaceName> ...`

*Example*

ectool modifyWorkspace test --description "my test workspace"

**Back to Top**

**resolveFile**

Resolves the path to a log file or artifact in a workspace.

You must specify `fromAgentId` and `workspaceName`. 
Arguments  |  Descriptions  
--- | ---  
fromAgentId  | Identifier of the agent requesting the route to a destination agent or artifact repository.  
workspaceName  | Name of the workspace.  
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.  
jobStepId  | The unique identifier for a job step, assigned automatically when the job is created. Also accepts a job name assigned to the job by its name template.  
resourceName  | Name of the resource.  

Positional arguments
  fromAgentId, workspaceName

Response
  None or a status OK message.

ec-perl
  syntax: $cmdr->resolveFile ( <fromAgentId>, <workspaceName>, {optionals}>);  
  Example
    $cmdr->resolveFile ("Machine2", "Dev_WS", {resourceName=> Server1});

ectool
  syntax: ectool resolveFile <fromAgentId> <workspaceName> [optionals ...]  
  Example
    ectool resolveFile "Machine2" "Dev_WS" --resourceName Server1

Back to Top

API Commands - Miscellaneous Management

  acquireNamedLock on page 650  
  changeOwner on page 651  
  clone on page 654  
  countObjects on page 661  
  deleteObjects on page 664  
  dumpHeap on page 668  
  dumpStatistics on page 668
**acquireNamedLock**

Gets the named lock.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>lockName</td>
<td>Name of the lock.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>create</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>When this argument is set to true or 1, the system will create a lock if it does not exist.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
</tbody>
</table>

**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**

```excel
ectool acquireNamedLock "Group 2" true
```

---

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>(Optional) The name of the application. Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) The name of the application tier. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) The name of the component. Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) The name of the email configuration. Argument type: String</td>
</tr>
<tr>
<td>credentialName</td>
<td>(Optional) <code>credentialName</code> can be in one of these formats:</td>
</tr>
<tr>
<td></td>
<td>- <strong>relative</strong> (for example, &quot;cred1&quot;) - the credential is assumed to be in</td>
</tr>
<tr>
<td></td>
<td>the project that contains the request target object. Requires a qualifying</td>
</tr>
<tr>
<td></td>
<td>project name.</td>
</tr>
<tr>
<td></td>
<td>- <strong>absolute</strong> (for example, &quot;projects/BuildProject/credentials/cred1&quot;) -</td>
</tr>
<tr>
<td></td>
<td>the credential can be from any specified project, regardless of the target</td>
</tr>
<tr>
<td></td>
<td>object’s project.</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) The name of an environment. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) The name of an environment template. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| environmentTierName | (Optional) The name of an environment tier.  
Argument type: String                                                                                                                        |
| groupName          | (Optional) The full name of a group. For Active Directory and LDAP, this is a full domain name.  
Argument type: String                                                                                                                                 |
| newOwnerName       | (Optional) The name of the new owner for this object. This defaults to the current user.  
Argument type: String                                                                                                                                 |
| notifierName       | (Optional) The name of the email notifier.  
Argument type: String                                                                                                                        |
| pluginName         | (Optional) The name of the plugin. This is the plugin key for a promoted plugin or a plugin key and version for an unpromoted plugin.  
Argument type: String                                                                                                                        |
| procedureName      | (Optional) The name of the procedure. It can be a path to the procedure. When using this argument, you must also enter the projectName.  
Argument type: String                                                                                                                        |
| processName        | (Optional) The name of a process. It can be a path to the process.  
Argument type: String                                                                                                                        |
| processStepName    | (Optional) The name of a process step. It can be a path to the process step.  
Argument type: String                                                                                                                        |
| projectName        | (Optional) The name of the project. It can be a path to the project. The project name is ignored for credentials, procedure, steps, and schedules when it is specified as a path.  
Argument type: String                                                                                                                        |
| propertySheetId    | (Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created.  
Argument type: UUID                                                                                                                         |
| resourceName       | (Optional) The name of the resource.  
Argument type: String                                                                                                                        |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule. It can be a path to the schedule. When using this argument, you must also use projectName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) The name of the step. It can be a path to the step. When using this argument, you must also enter projectName and procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The full name of the user. For Active Directory and LDAP, the name can be user@domain.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

None

**Response**

Returns the modified object.

**ec-perl**

*Syntax:* $cmdr->changeOwner({...});

*Example*

$cmdr->changeOwner ("projectName":"Sample Project");

**ectool**

*Syntax:* ectool changeOwner ...

*Example*

ectool changeOwner --projectName "Sample Project"

Back to Top
**clone**

Makes a copy of an existing ElectricFlow object. For example: credential, directory provider, email configuration, email notifier, project, procedure, property sheet, resource, resource pool, schedule, state definition, step, transition definition, workflow definition, and workspace.

**Note:** You cannot clone parameters.

**IMPORTANT:**
To find the entity you want to clone, you must specify the following arguments:
- A new name for the cloned object (cloneName)
- Locator arguments

For example, if you want to clone a project, you must specify the name of the project that you want to clone.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Naming</strong></td>
<td></td>
</tr>
<tr>
<td>cloneName</td>
<td><em>(Optional)</em> The cloneName specifies the path to the new object, possibly in an alternate location. If no container path is specified, the new object is created inside the same container as the original. If no name is specified, the server will generate a name.</td>
</tr>
<tr>
<td><strong>Optional</strong></td>
<td></td>
</tr>
<tr>
<td>disableProjectTracking</td>
<td><em>(Optional)</em> &lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
</table>
| reducedDetailChangeHistory| (Optional) <Boolean flag - 0|1|true|false>  
Use this argument for large projects containing over 20,000 audited objects with Change Tracking enabled.  
When this argument is set to true or 1, ElectricFlow automatically decreases the amount of Change History indexing information that it saves in a large project, reducing the level of detail for Change Tracking-intensive operations in the Change History. This can make it harder to revert an object to a specific state and to find information in the Change History when you are troubleshooting or debugging an issue.  
Set this argument to false or 0 to suppress this behavior so that ElectricFlow does not change the amount of indexing information for a large project. This will cause the operation to take longer and put more load on the database, but the Change History will have the full details of the entities owned by objects in the project.  
Argument type: Boolean |

### Locators

<table>
<thead>
<tr>
<th>Locator</th>
<th>Description</th>
</tr>
</thead>
</table>
| applicationName         | (Optional) The name of the application that is 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.  
Argument type: String |
| cloneName               | (Optional) The new name of the cloned copy of an object.  
Argument type: String |
| componentName           | (Optional) The name of the component.  
Argument type: String |
| configName              | (Optional) The name of the email configuration.  
Argument type: String |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td><em>(Optional)</em> The name of the credential that can be specified in one of these formats:</td>
</tr>
<tr>
<td></td>
<td><strong>relative</strong> <em>(for example, &quot;cred1&quot;)—The credential is assumed to be in the project that contains the target object.</em></td>
</tr>
<tr>
<td></td>
<td><strong>absolute</strong> <em>(for example, &quot;/projects/BuildProject/credentials/cred1&quot;)—The credential can be from any specified project, regardless of the project with the target object.</em></td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentName</td>
<td><em>(Optional)</em> The name of the environment that must be unique among all projects.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td><em>(Optional)</em> The name of the environment template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td><em>(Optional)</em> The name of the environment template tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td><em>(Optional)</em> The name of the environment tier.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>flowName</td>
<td><em>(Optional)</em> Name of the flow that must be unique within the project.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>flowRuntimeName</td>
<td><em>(Optional)</em> Name of the flow runtime.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>flowRuntimeStateName</td>
<td><em>(Optional)</em> Name of the flow state.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>flowStateName</td>
<td><em>(Optional)</em> Name of the flow state that must be unique within the flow.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>flowTransitionName</td>
<td>Name of the flow transition that must be unique within the flow state.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: String</td>
</tr>
<tr>
<td>gatewayName</td>
<td><em>(Optional)</em> The name of the gateway.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of the group. Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>(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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) The object id as returned by <code>findObjects</code>. Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) The property path for the object. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(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</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project that you want to clone. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>providerName</strong></td>
<td>(Optional) The unique name of the directory provider, such as the LDAP or Active Directory provider name.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>releaseName</strong></td>
<td>(Optional) The name of the release which owns the property.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>repositoryName</strong></td>
<td>(Optional) The name of the repository used for artifact management.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>resourceName</strong></td>
<td>(Optional) The name of the resource that you want to clone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>resourcePoolName</strong></td>
<td>(Optional) The name of a pool containing one or more resources.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>resourceTemplateName</strong></td>
<td>(Optional) The name of the resource template.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>scheduleName</strong></td>
<td>(Optional) The name of the schedule that you want to clone. When using this argument, you must also enter projectName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>snapshotName</strong></td>
<td>(Optional) The name of the snapshot that you want to clone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>stageName</strong></td>
<td>(Optional) The name of the stage definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>stateDefinitionName</strong></td>
<td>(Optional) The name of the state definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>stateName</strong></td>
<td>(Optional) The name of the state.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td><strong>stepName</strong></td>
<td>(Optional) The name of the step that you want to clone. When using this argument, you must also enter projectName and procedureName.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) System object names include: admin</td>
</tr>
<tr>
<td></td>
<td>Argument type: SystemObjectName</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) The name of the transition definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) The name of the transition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) The name of the user where you may need to expand the string.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) The name of the workflow definition.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) The name of the workflow.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) The name of the workspace that you want to clone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) The name of the zone.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

None.

**Response**

Returns the name of the new cloned object.

Using the `clone` command successfully depends on the context of the locator arguments in your system. The command works when the arguments are specified correctly.

**ec-perl**

*Syntax:* $cmdr->clone ...;
**Examples**

# Create a copy of a procedure, as though you clicked the "Copy" button in the UI.

```xo
$XPath = $cmdr->clone(
   
   projectName => "EC-Examples",
   procedureName => "set Property"
);
```

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

```xo
$XPath = $cmdr->clone(
   
   projectName => "EC-Examples",
   procedureName => "set Property",
   cloneName => "set Property 2"
);
```

# Create a copy of a procedure step.

```xo
$XPath = $cmdr->clone(
   
   projectName => "EC-Examples",
   procedureName => "set Property",
   cloneName => "set Property 2",
   stepName => 'setProperty'
);
```

# Copy a step using the path.

```xo
$XPath = $cmdr->clone(
   
   path =>
      '/projects/EC-Examples/procedures/set Property/steps/setProperty'
);
```

**ectool**

*syntax:* ectool clone ...

**Examples**

# Create a copy of a procedure, as though you clicked the "Copy" button in the UI.

```xo
$ 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.
countObjects

Returns the count of objects specified by the provided filter.

You must enter objectType.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectType</td>
<td>The type of object to count. Values include:</td>
</tr>
<tr>
<td></td>
<td>application</td>
</tr>
<tr>
<td></td>
<td>applicationTier</td>
</tr>
<tr>
<td></td>
<td>artifact</td>
</tr>
<tr>
<td></td>
<td>artifactVersion</td>
</tr>
<tr>
<td></td>
<td>component</td>
</tr>
<tr>
<td></td>
<td>credential</td>
</tr>
<tr>
<td></td>
<td>deployerApplication</td>
</tr>
<tr>
<td></td>
<td>deployerConfiguration</td>
</tr>
<tr>
<td></td>
<td>directoryProvider</td>
</tr>
<tr>
<td></td>
<td>emailconfig</td>
</tr>
<tr>
<td></td>
<td>emailNotifier</td>
</tr>
<tr>
<td></td>
<td>environment</td>
</tr>
<tr>
<td></td>
<td>environmentTemplate</td>
</tr>
<tr>
<td></td>
<td>environmentTemplateTier</td>
</tr>
<tr>
<td></td>
<td>environmentTier</td>
</tr>
<tr>
<td></td>
<td>formalParameter</td>
</tr>
<tr>
<td></td>
<td>job</td>
</tr>
<tr>
<td></td>
<td>jobStep</td>
</tr>
<tr>
<td></td>
<td>logEntry</td>
</tr>
<tr>
<td></td>
<td>pipeline</td>
</tr>
<tr>
<td></td>
<td>plugin</td>
</tr>
<tr>
<td></td>
<td>procedure</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td></td>
<td>procedureStep</td>
</tr>
<tr>
<td></td>
<td>processDependency</td>
</tr>
<tr>
<td></td>
<td>processStep</td>
</tr>
<tr>
<td></td>
<td>project</td>
</tr>
<tr>
<td></td>
<td>property</td>
</tr>
<tr>
<td></td>
<td>release</td>
</tr>
<tr>
<td></td>
<td>repository</td>
</tr>
<tr>
<td></td>
<td>resource</td>
</tr>
<tr>
<td></td>
<td>resourcePool</td>
</tr>
<tr>
<td></td>
<td>resourceTemplate</td>
</tr>
<tr>
<td></td>
<td>schedule</td>
</tr>
<tr>
<td></td>
<td>snapshot</td>
</tr>
<tr>
<td></td>
<td>stage</td>
</tr>
<tr>
<td></td>
<td>state</td>
</tr>
<tr>
<td></td>
<td>stateDefinition</td>
</tr>
<tr>
<td></td>
<td>task</td>
</tr>
<tr>
<td></td>
<td>tierMap</td>
</tr>
<tr>
<td></td>
<td>transition</td>
</tr>
<tr>
<td></td>
<td>transitionDefinition</td>
</tr>
<tr>
<td></td>
<td>workflowDefinition</td>
</tr>
<tr>
<td></td>
<td>workspace</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>filter</td>
<td>(Optional) A list of zero or more filter criteria definitions used to define objects to find. Each element of the filter list is a hash reference containing one filter criterion. You can specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricFlow Perl API. Two types of filters: &quot;property filters&quot; - used to select objects based on the value of the object's intrinsic or custom property &quot;boolean filters&quot; (&quot;and&quot;, &quot;or&quot;, &quot;not&quot;) - used to combine one or more filters using boolean logic. Each &quot;property filter&quot; consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by ElectricFlow or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property. Property filter operators are:</td>
</tr>
<tr>
<td></td>
<td>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)</td>
</tr>
<tr>
<td></td>
<td>A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter. Boolean operators are:</td>
</tr>
<tr>
<td></td>
<td>not (1 operand) and (2 or more operands) or (2 or more operands)</td>
</tr>
</tbody>
</table>

**Argument type: Collection**

**Positional arguments**

objectType
**Response**

Returns the number of filtered objects.

**ec-perl**

*syntax:* $cmdr->countObjects(<objectType>, [<optionals>]);

**Example**

use ElectricCommander();
my @artifactNameFilters;
# Create the filter list for filtering on artifact name
push (@artifactNameFilters,
    {"propertyName"=>"artifactName",
     "operator"=>"contains",
     "operand1"=>"groupId:installer-windows",
    });
my $cmdr = new ElectricCommander();
# Perform the countObjects query
my $reference=$cmdr->countObjects("artifactVersion",
    { filter=>
      {operator=>"and",
       filter=>[
         { propertyName=>"modifyTime",
           operator=>"greaterOrEqual",# Give me all dates after or equal arbitrary date
           operand1=>"2014-03-25T14:48:55.286Z",
         },
         { operator => 'or', # apply 'or' for the filters in the list
          filter => \@artifactNameFilter
        ]
      }
    });

my $jobs=$reference->find('//response/count');
print $jobs;

**ectool**

Not supported.

**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`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>objectType</code></td>
<td>This argument specifies the type of object to find. Values include: `artifact</td>
</tr>
</tbody>
</table>
Arguments | Descriptions
--- | ---

(Optional) Specify filters in a space-separated list: filter1 filter2 ...
A list of zero or more filter criteria definitions used to define objects to find.

Each element of the filter list is a hash reference containing one filter criterion. You may specify several filter criteria, in which case an object must meet all filter criteria to be included in the result. See the code example below for instructions on forming the list and passing it to the ElectricFlow Perl API.

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

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

Property filter operators are:

- between (2 operands)
- contains (1)
- equals (1)
- greaterOrEqual (1)
- greaterThan (1)
- in (1)
- lessOrEqual(1)
- lessThan (1)
- like (1)
- notEqual (1)
- notLike (1)
- isNotNull (0)
- isNull (0)

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

Boolean operators are:

- not (1 operand)
- and (2 or more operands)
- or (2 or more operands)

Argument type: Collection
ElectricFlow Perl API Commands

Arguments

Descriptions

maxIds

(Optional) <id count>
The maximum number of objects that will be deleted. The default
is all objects that match the filter.
Argument type: Integer
(Optional) Specify "sorts" in a space-separated list: sort1 sort2
...
An ordered list of sort criteria. Each list entry consists of a property
name and a sort order--either an ascending or descending sort
order.
If you specify more than one sort criterion, the sorts are applied
according to the order they appear in the list.

sorts

The first item in the list is the primary sort key.
Each item in the list is a hash reference.
See the code example below for instructions about forming the list
and passing it to the ElectricFlow Perl API.
The sort order affects which objects are deleted if a maxIds value
limits the number of objects returned by the filter.
Argument type: Collection

Positional arguments
objectType

Response
Returns a list of object references.

ec-perl
syntax: $cmdr->deleteObjects(<objectType>, {<optionals>});

Example
This code example illustrates using a Boolean filter for the deleteObjects command to find jobs matching
either of two patterns for the job name.

my @filterList;
push (@filterList, {"propertyName" => "jobName",
"operator" => "like",
"operand1" => "%-branch-%"});
push (@filterList, {"propertyName" => "jobName",
"operator" => "like",
"operand1" => "branch-%"});
my $result = $cmdr->deleteObjects('job',
{filter => [
{ operator => 'or',
filter => \@filterList,
}

667


```
}
});
print "result = " . $result->findnodes_as_string("n"). "\n";
```

**ectool**

Not supported.

**dumpHeap**

Captures a Java heap dump.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>Name of the file to which the heap dump is written.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

**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]'

**dumpStatistics**

Prints (emits) internal timing statistics.
Arguments | Descriptions
--- | ---
clearStatistics | (Optional) `<Boolean flag> - 0|1|true|false>
If this argument is set to true, the system clears the statistics after logging them.
Argument type: Boolean
dumpLapTimes | (Optional) `<Boolean flag> - 0|1|true|false>
If this argument is set to true, the system dumps lap times.
If this argument is set to false, the system dumps global times.
Argument type: Boolean
fileName | (Optional) If you specify a file name with a path, the output is redirected to this file.
When the path is relative, the file is written relative to the working directory of the server.
Argument type: String
format | Format of the output.
Valid values are text or xml.
The default is text.
Argument type: String

Positional arguments
None

Response
None or a status OK message.

ec-perl

`$cmdr->dumpStatistics {{<optionals>}};`

*Example*

```
$cmdr->dumpStatistics {{clearStatistics=>true, format=>xml}};
```

ectool

`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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dsl</code></td>
<td>The DSL text. Argument type: String</td>
</tr>
<tr>
<td><code>applicationName</code></td>
<td>(Optional) Name of the application. Argument type: String</td>
</tr>
<tr>
<td><code>applicationTierName</code></td>
<td>(Optional) Name of the application tier. Argument type: String</td>
</tr>
<tr>
<td><code>artifactName</code></td>
<td>(Optional) Name of the artifact. Argument type: String</td>
</tr>
<tr>
<td><code>artifactVersionName</code></td>
<td>(Optional) Name of the artifact version. Argument type: String</td>
</tr>
<tr>
<td><code>componentName</code></td>
<td>(Optional) Name of the component. Argument type: String</td>
</tr>
<tr>
<td><code>configName</code></td>
<td>(Optional) Name of the email configuration. Argument type: String</td>
</tr>
<tr>
<td><code>credentialName</code></td>
<td>(Optional) Name of the credential. Argument type: String</td>
</tr>
<tr>
<td>debug</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td>describe</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>environmentName</td>
<td>(Optional) Name of the environment. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>(Optional) Name of the environment template. Argument type: String</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>(Optional) Name of the environment template tier. Argument type: String</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>(Optional) Name of the environment tier. Argument type: String</td>
</tr>
<tr>
<td>gatewayName</td>
<td>(Optional) The name of the gateway. Argument type: String</td>
</tr>
<tr>
<td>groupName</td>
<td>(Optional) The name of the group. Argument type: String</td>
</tr>
<tr>
<td>jobId</td>
<td>(Optional) The primary key or name of the job. Argument type: UUID</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The primary key or name of the job step. Argument type: UUID</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) The name of the notifier. Argument type: UUID</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) The object ID returned by the findObjects call. Argument type: UUID</td>
</tr>
<tr>
<td>parameters</td>
<td>(Optional) Parameters are passed to the script by ElectricFlow as JSON text. Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) The property path. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) The name of the plugin. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) The name of the procedure. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) The name of the process if the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) The name of the process step if the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) The name of the project. Argument type: String</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The primary key or name of the property sheet. Argument type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) The name of the repository. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) The name of the resource. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) The name of the resource pool. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) The name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) The name of the schedule. Argument type: String</td>
</tr>
<tr>
<td>serverLibraryPath</td>
<td>(Optional) Path to the server directory that contains jar files and classes to be added to the classpath when evaluating the DSL text. Argument type: String</td>
</tr>
</tbody>
</table>

**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

**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` to evaluate.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Script to evaluate in the specified context. Argument type: String</td>
</tr>
<tr>
<td>applicationName</td>
<td>(Optional) Name of the application that must be unique among all projects. Argument type: String</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>(Optional) Name of the application tier. Argument type: String</td>
</tr>
<tr>
<td>artifactName</td>
<td>(Optional) Name of the artifact. Argument type: String</td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactVersionName</td>
<td>(Optional) Name of the artifact version. <strong>Note:</strong> An artifact version name is interpreted by the server as the <code>artifactVersionName</code> attribute for the <code>artifactVersion</code> in question. This name is parsed and interpreted as &quot;groupId:artifactKey:version&quot; and the object is searched either way you specify its name. The ElectricFlow server interprets either name form correctly. Argument type: String</td>
</tr>
<tr>
<td>componentName</td>
<td>(Optional) Name of the component. Argument type: String</td>
</tr>
<tr>
<td>configName</td>
<td>(Optional) Name of the email configuration. Argument type: String</td>
</tr>
</tbody>
</table>
| credentialName      | (Optional) Name of the credential in one of these forms:  
  - **relative** (for example, "cred1")—The credential is assumed to be in the project that contains the request target object.  
  - **absolute** (for example, "/projects/BuildProject/credentials/cred1")—The credential can be from any specified project, regardless of the target object’s project. Argument type: String |
<p>| environmentName     | (Optional) Name of the environment that must be unique among all projects. Argument type: String                                                                                                                                                                                                                                                                                                         |
| environmentTemplateName | (Optional) Name of the environment template. Argument type: String                                                                                                                                                                                                                                                                                                                                         |
| environmentTemplateTierName | (Optional) Name of the environment template tier. Argument type: String                                                                                                                                                                                                                                                                                                                          |
| environmentTierName | (Optional) Name of the environment tier. Argument type: String                                                                                                                                                                                                                                                                                                                                          |
| gatewayName         | (Optional) Name of the gateway. Argument type: String                                                                                                                                                                                                                                                                                                                                                       |
| groupName           | (Optional) Name of a group where you might evaluate a script. Argument type: String                                                                                                                                                                                                                                                                                                                         |</p>
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>(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</td>
</tr>
<tr>
<td>jobStepId</td>
<td>(Optional) The unique identifier for a job step that is assigned automatically when the job step is created. Argument type: UUID</td>
</tr>
<tr>
<td>notifierName</td>
<td>(Optional) Name of the email notifier. Argument type: String</td>
</tr>
<tr>
<td>objectId</td>
<td>(Optional) This is an object identifier returned by <code>findObjects</code> and <code>getObjects</code>. Argument type: String</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) Property path string. Argument type: String</td>
</tr>
<tr>
<td>pipelineName</td>
<td>(Optional) The name of the pipeline. Argument type: String</td>
</tr>
<tr>
<td>pluginName</td>
<td>(Optional) Name of a plugin where you might evaluate a script. Argument type: String</td>
</tr>
<tr>
<td>procedureName</td>
<td>(Optional) Name of a procedure where you might need to evaluate a script. <strong>Also requires</strong> <code>projectName</code>. Argument type: String</td>
</tr>
<tr>
<td>processName</td>
<td>(Optional) Name of the process when the container is a process or process step. Argument type: String</td>
</tr>
<tr>
<td>processStepName</td>
<td>(Optional) Name of the process step when the container is a process step. Argument type: String</td>
</tr>
<tr>
<td>projectName</td>
<td>(Optional) Name of the project that contains the script to evaluate. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>(Optional) The unique identifier for a property sheet that is assigned automatically when the property sheet is created. Argument type: UUID</td>
</tr>
<tr>
<td>releaseName</td>
<td>(Optional) The name of the release container of the property sheet which owns the property. Argument type: String</td>
</tr>
<tr>
<td>repositoryName</td>
<td>(Optional) Name of the repository for artifact management. Argument type: String</td>
</tr>
<tr>
<td>resourceName</td>
<td>(Optional) Name of a resource where you might evaluate a script. Argument type: String</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>(Optional) Name of a pool containing one or more resources. Argument type: String</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>(Optional) Name of the resource template. Argument type: String</td>
</tr>
<tr>
<td>scheduleName</td>
<td>(Optional) Name of a schedule within this project. Also requires projectName. Argument type: String</td>
</tr>
<tr>
<td>snapshotName</td>
<td>(Optional) Name of the snapshot. Argument type: String</td>
</tr>
<tr>
<td>stageName</td>
<td>(Optional) The name of the stage definition. Argument type: String</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>(Optional) Name of the state definition. Argument type: String</td>
</tr>
<tr>
<td>stateName</td>
<td>(Optional) Name of the state. Argument type: String</td>
</tr>
<tr>
<td>stepName</td>
<td>(Optional) Name of the step whose script you might evaluate. Also requires projectName and procedureName. Argument type: String</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>(Optional) System object names include: admin</td>
</tr>
<tr>
<td>taskName</td>
<td>(Optional) The name of the task. Argument type: String</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>(Optional) Name of the transition definition. Argument type: String</td>
</tr>
<tr>
<td>transitionName</td>
<td>(Optional) Name of the transition. Argument type: String</td>
</tr>
<tr>
<td>userName</td>
<td>(Optional) Name of the user where you may need to evaluate a script. Argument type: String</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>(Optional) Name of the workflow definition. Argument type: String</td>
</tr>
<tr>
<td>workflowName</td>
<td>(Optional) Name of the workflow. Argument type: String</td>
</tr>
<tr>
<td>workspaceName</td>
<td>(Optional) Name of a workspace where you may need to evaluate a script. Argument type: String</td>
</tr>
<tr>
<td>zoneName</td>
<td>(Optional) Name of the zone. Argument type: String</td>
</tr>
</tbody>
</table>

**Positional arguments**

value

**Response**

The string value of the final expression in the Javascript block inside a value element.

**ec-perl**

*Syntax:* $cmdr->evalScript (<value>);

**Examples**

my $result = $ec->evalScript (q{"ip=" + server.hostIP+", name=" + server.hostName}) ->findvalue("/value");
my $result = $ec->evalScript (q\{myProject.projectName}, {jobId => '4fa765dd-73f1-11e3-b67e-b0a420524153'});

ectool

**syntax:** ectool evalScript <value>

**Examples**

ectool evalScript '"ip=" + server.hostIP", name=" + server.hostName'

ectool evalScript 'myProject.projectName' --jobId 4fa765dd-73f1-11e3-b67e-b0a420524153

--jobStepId 5da765dd-73f1-11e3-b67e-b0a420524153

**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.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td><code>&lt;remoteFileName&gt;</code> The specified directory for the file must already exist in the system. If the path is local, it will be created on the server. If it is a network path, it must be accessible by the server and the server user. Argument type: String</td>
</tr>
<tr>
<td>compress</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| disableProjectTracking     | (Optional) `<Boolean flag - 0|1|true|false>`  
If set to `true` when importing or exporting a project, even if the original project had Change Tracking enabled, make Change Tracking of the newly imported or exported project be disabled from its creation.  
If you do not need to track changes to the new project, this avoids the Change Tracking overhead that would otherwise slow down the import operation, and also saves having to subsequently disable change tracking of the re-imported project.  
Argument type: Boolean      |
| download                   | (Optional) `<Boolean flag - 0|1|true|false>`  
If set to 1 or `true`, the exported file can be downloaded.  
Argument type: Boolean      |
| excludeJobs                | (Optional) `<Boolean flag - 0|1|true|false>`  
If set to 1, no job information will be exported. This argument can be used to reduce the size of the export file.  
Argument type: Boolean      |
| objectId                   | (Optional) ID of the object ID.  
Argument type: UUID         |
| path                       | (Optional) `<property path>`  
Specifies the path relative to the server process for an object to be exported. Any single object can be exported if it is specified using property path syntax. The object and its subobjects are exported.  
Argument type: String       |
| reducedDetailChangeHistory | (Optional) `<Boolean flag - 0|1|true|false>`  
Use this argument for large projects containing over 20,000 audited objects with Change Tracking enabled.  
When this argument is set to `true` or `1`, ElectricFlow automatically decreases the amount of Change History indexing information that it saves in a large project, reducing the level of detail for Change Tracking-intensive operations in the Change History. This can make it harder to revert an object to a specific state and to find information in the Change History when you are troubleshooting or debugging an issue.  
Set this argument to `false` or `0` to suppress this behavior so that ElectricFlow does not change the amount of indexing information for a large project. This will cause the operation to take longer and put more load on the database, but the Change History will have the full details of the entities owned by objects in the project.  
Argument type: Boolean      |
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| relocatable          | (Optional) `<Boolean flag - 0|1|true|false>` If the `--relocatable` flag is set to `true`, a partial export (for example, with `--path`) will not include object IDs, ACLs, system property sheets, create/modify times, owners, email notifiers or `lastModifiedBy` information, and the export file result will be much smaller than a normal export. When this file is imported, the result should show one or more objects owned by the importing user as if they were newly created.  
**Note:** The relocatable argument only works with a partial export. This argument is silently ignored during a full export.  
Argument type: Boolean |
| revisionNumber       | (Optional) Revision number of the file.  
Argument type: Integer                                                                                                                |
| safeMode             | (Optional) The `safeMode` argument determines whether the server will be quiesced before a full export begins and if yes, whether or not the server will shutdown and restarted after the export completes. Values are:  
- none (default) - Do not quiesce the server during export.  
- shutdown - Quiesce the server and shutdown when complete.  
- restart - Quiesce the server and restart when complete.  
**Note:** The `safeMode` argument has no effect on partial exports.  
Argument type: SafeMode |
| withAcls             | (Optional) This argument modifies `relocatable`.  
<`Boolean flag - 0|1|true|false>` If the `withAcls` flag is set to "true", a relocatable partial export will include ACLs.  
Argument type: Boolean |
| withNotifiers        | (Optional) This argument modifies `relocatable`.  
<`Boolean flag - 0|1|true|false`>  
If this flag is set to `true`, a relocatable partial export will include email notifiers.  
Argument type: Boolean |
| withVersionNumbers   | (Optional) `<Boolean flag - 0|1|true|false>`  
If this flag is set to `true`, version numbers are issued during the export operation (they are ignored on import). |

**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> ...

**Examples**
ectool export "c:\ElectricCommanderBackup\Aug 15 2015.xml"

ectool export "c:\ElectricCommanderBackup\Test Proj.xml" --path "/projects[Test Proj]"
--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.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectType</td>
<td>The type of object to find.</td>
</tr>
<tr>
<td></td>
<td><strong>Values include:</strong></td>
</tr>
<tr>
<td></td>
<td>application</td>
</tr>
<tr>
<td></td>
<td>artifact</td>
</tr>
<tr>
<td></td>
<td>artifactVersion</td>
</tr>
<tr>
<td></td>
<td>component</td>
</tr>
<tr>
<td></td>
<td>credential</td>
</tr>
<tr>
<td></td>
<td>directoryProvider</td>
</tr>
<tr>
<td></td>
<td>emailconfig</td>
</tr>
<tr>
<td></td>
<td>emailNotifier</td>
</tr>
<tr>
<td></td>
<td>environment</td>
</tr>
<tr>
<td></td>
<td>formalParameter</td>
</tr>
<tr>
<td></td>
<td>job</td>
</tr>
<tr>
<td></td>
<td>jobStep</td>
</tr>
<tr>
<td></td>
<td>logEntry</td>
</tr>
<tr>
<td></td>
<td>plugin</td>
</tr>
<tr>
<td></td>
<td>procedure</td>
</tr>
<tr>
<td></td>
<td>procedureStep</td>
</tr>
<tr>
<td>Argument type: String</td>
<td></td>
</tr>
</tbody>
</table>
### Arguments

<table>
<thead>
<tr>
<th>filters</th>
<th>(Optional) A list of zero or more filter criteria definitions used to define objects to find.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
|         | Two types of filters:  
|         | "property filters" - used to select objects based on the value of the object's intrinsic or custom property  
|         | "boolean filters" ("and", "or", "not") - used to combine one or more filters using boolean logic. |
|         | Each "property filter" consists of a property name to test and an operator to use for comparison. The property can be either an intrinsic property defined by ElectricFlow or a custom property added by the user. Each operator takes zero, one, or two operands to compare against the desired property. |
|         | Property filter operators are:  
|         | between (2 operands)  
|         | contains (1)  
|         | equals (1)  
|         | greaterOrEqual (1)  
|         | greaterThan (1)  
|         | in (1)  
|         | lessOrEqual (1)  
|         | lessThan (1)  
|         | like (1)  
|         | notEqual (1)  
|         | notLike (1)  
|         | isNotNull (0)  
|         | isNull (0)  
|         | A boolean filter is a boolean operator and an array of one or more filters that are operands. Each operand can be either a property filter or a boolean filter. |
|         | Boolean operators are:  
|         | not (1 operand)  
|         | and (2 or more operands)  
|         | or (2 or more operands)  
|         | Argument type: Collection |
| includeAccess | (Optional) `<Boolean flag> 0|1|true|false>` If set to true, this command also returns access maps for the objects. |
| Argument type: Boolean |
### Arguments

<table>
<thead>
<tr>
<th><strong>Arguments</strong></th>
<th><strong>Descriptions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>includeLatestRevision</code></td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td><code>maxIds</code></td>
<td>(Optional) <code>&lt;id count&gt;</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 &quot;0&quot; is specified, the ID of every matching object is returned. Argument type: Integer</td>
</tr>
<tr>
<td><code>numObjects</code></td>
<td>(Optional) <code>&lt;full object count&gt;</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 &quot;full objects&quot; 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. Argument type: Integer</td>
</tr>
<tr>
<td><code>selects</code></td>
<td>(Optional) This is an unordered list of property names that specify additional top-level properties to return for each object. See the code example below for instructions on forming the list and passing it to the ElectricFlow Perl API. Argument type: Collection</td>
</tr>
<tr>
<td><code>sorts</code></td>
<td>(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</td>
</tr>
</tbody>
</table>

### Positional arguments

| **objectType** | |

### Response

This command returns a list of object references. These references can be used in a subsequent call to the `getObjects` command. 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.

```perl
my @filterList;
push (@filterList, {"propertyName" => "jobName", "operator" => "like", "operand1" => ":%branch-%"});
push (@filterList, {"propertyName" => "jobName", "operator" => "like", "operand1" => "%branch-%"});
my $result = $cmdr->findObjects('job', {filter => [@filterList],});
print "result = " . $result->findnodes_as_string("/"). ":\n";
```

**Example 2**
This example uses `findObjects` and `getObjects` to manage large result sets, and also uses `select` to return the values of two properties in the returned objects.

```perl
# Search for the first 10 matching objects and retrieve the first 2
my $XPath = $cmdr->findObjects("schedule", {maxIds => "10", numObjects => "2", filter => [{propertyName => "createTime", operator => "greaterOrEqual", operand1 => "2007-01-20T00:00:00.000Z"}, {propertyName => "lastModifiedBy", operator => "like", operand1 => "adm%"}], sort => [{propertyName => "projectName", order => "ascending"}, {propertyName => "createTime", order => "descending"}], select => [{propertyName => 'prop1'}, {propertyName => 'prop2'}]});
print "Return data from 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];
```

```
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.

```perl
# 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'}]});
```

tectool
Not supported.

Back to Top

finishCommand
The agent uses this command to indicate that a command has been run.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>agentRequestId</td>
<td>Request ID of the command.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>status</td>
<td>Status of the command.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>deletedLogFile</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>deletedPostpLogFile</td>
<td>(Optional) &lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>elapsedTime</td>
<td>(Optional) Elapsed time that the command runs.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Double</td>
</tr>
<tr>
<td>errorMessage</td>
<td>(Optional) Error message from the agent.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>exit</td>
<td>(Optional) Exit code of the command.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>maxProtocolVersion</td>
<td>(Optional) Maximum protocol version that the agent supports.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>minProtocolVersion</td>
<td>(Optional) Minimum protocol version that the agent supports.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>pingToken</td>
<td>(Optional) Current ping token.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Long</td>
</tr>
<tr>
<td>postpExit</td>
<td>(Optional) Exit code of postp.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>protocolVersion</td>
<td>(Optional) Current agent protocol version.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Integer</td>
</tr>
<tr>
<td>version</td>
<td>(Optional) Current agent version.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>
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 enter the dsl argument.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>A property path indicating a single object for which DSL will be generated</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
</tbody>
</table>

Positional arguments
path

Response
Content for the DSL script. For an example, go to Getting Started with DSL on page 760.

ec-perl
syntax:$cmdr->generateDsl(<path>);

Example
$cmdr->generateDsl(/resources/local);

ectool
syntax:ectool generateDsl <path>
**Example**

`ectool generateDsl /resources/local`

**getObjects**

The `getObjects` command retrieves a list of full objects based on object IDs returned by `findJobSteps` or `findObjects`. All requested objects must be of the same `objectType`. See `findObjects` for a list of object types.

You must specify `objectIds`.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>objectIds</code></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Argument Type: Collection</td>
</tr>
<tr>
<td><code>includeAccess</code></td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td><code>includeLatestRevision</code></td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td><code>selects</code></td>
<td>(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.</td>
</tr>
<tr>
<td></td>
<td>Argument type: Collection</td>
</tr>
</tbody>
</table>

**Positional arguments**

`objectIds`

**Response**

A list of full objects for the requested type.

**ec-perl**

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

**Example 1**

Code example for `findObjects` and `getObjects`:
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 = $H->findObjects('project', {
  filter => {operator => 'and',
              filter => [{
    propertyName => 'projectName',
    operator => 'contains',
    operand1 => $projectBase},
    {propertyName => 'description',
    operator => 'equals',
    operand1 => 'foo'}]}]);
**graphStateMachine**

Generates a graph element with a state machine DOT graph as CDATA content.

You must specify a **jobId**.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>jobId</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job 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</td>
</tr>
</tbody>
</table>
| cluster   | (Optional) `<Boolean flag - 0|1|true|false>`  
If this argument is set to 1 or true, the graphs are clustered. Argument type: Boolean |

**Positional arguments**

**jobId**

**Response**

CDATA content.

**ec-perl**

*Syntax examples:* `$cmdr->graphStateMachine (<jobId>, {{<optionals>>});

*Examples*

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

**ectool**

*Syntax examples:* `ectool graphStateMachine <jobId> ...

*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.
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>fileName</td>
<td>Name of the file to import:</td>
</tr>
<tr>
<td></td>
<td>- <code>&lt;remoteFileName&gt;</code> This is the name of a file on the server to import. The file path name must be accessible to the server process on the server host.</td>
</tr>
<tr>
<td></td>
<td>- <code>&lt;localFileName&gt;</code> This is the path to a file on the client to import. The file is uploaded from the client to the server. The specified <code>&lt;file&gt;</code> value is sent as an attachment to the import API request.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>By default the limit is 50MB, so this option should be used only for individually exported objects, not a full system export.</td>
</tr>
<tr>
<td></td>
<td>Argument type: String</td>
</tr>
<tr>
<td>batchSize</td>
<td>(Optional) <code>&lt;batch size&gt;</code> The number of objects imported before committing a transaction in the database. This argument limits the object batch size during import. Default value is 50 objects. If your objects are unusually large, you can throttle this number down to 1, depending on your available memory.</td>
</tr>
<tr>
<td>disableProjectTracking</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>disableSchedules</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td></td>
<td>Argument type: Boolean</td>
</tr>
<tr>
<td>Arguments</td>
<td>Descriptions</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>force</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>path</td>
<td>(Optional) <code>&lt;property path&gt;</code> Use this argument to import a single object to a new location. For example, if a procedure was exported from &quot;project A&quot;, this argument allows you to import it into &quot;project B&quot;, but only if the export also used the <code>path</code> option. Argument type: String</td>
</tr>
<tr>
<td>preserveId</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>reducedDetailChangeHistory</td>
<td>(Optional) `&lt;Boolean flag - 0</td>
</tr>
</tbody>
</table>

**Positional arguments**

fileName

**Response**

None or a status OK message.

cmdr

**syntax examples:**

```perl
$cmdr->import(<fileName>, {...});
$cmdr->import({file => <localFileName>, ...});
```
Examples

```perl
$cmdr->import("/opt/TestProg.xml");

$cmdr->import({file => "c:\r.xml", path => "/projects[Test]"});
```

**ectool**

**syntax examples:** `ectool import <remoteFileName> ...`  
`ectool import <localFileName>`

**Examples**

`ectool import /mnt/backups/fullBackkup.xml`

`ectool "c:\project.xml" --path "/projects[Test]"`

Back to Top

---

**logStatistic**

Prints (emits) a statistics value to StatsD.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
</table>
| 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**

```perl
$cmdr->logStatistic("Interoperability performance test cases", "counters", 7);
```

**ectool**

**syntax:** `ectool logStatistic <name> <type> <value>`
**Example**

`ectool logStatistic "Interoperability performance test cases" "counters" 7`

---

**releaseNamedLock**

Releases the named lock that synchronizes the name of an object.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>lockName</td>
<td>Name of the lock. Argument type: String</td>
</tr>
<tr>
<td>delay</td>
<td>Number of seconds to delay releasing the lock. Argument type: Integer</td>
</tr>
</tbody>
</table>

**Positional arguments**

`lockName`

**Response**

None or a status OK message.

**ec-perl**

*Syntax examples:* `cmdr->releaseNamedLock (<lockName>, {<optionals>});`

*Examples*

```perl
$cmdr->releaseNamedLock ("group1", {delay => 5});
```

**ectool**

*Syntax examples:* `ectool releaseNamedLock <lockName> [optionals...]`

*Examples*

```bash
ectool releaseNamedLock "group1" --delay 5
```

Back to Top
API Response and Element Glossary

The first part of this topic lists returned response container elements in alphabetical order. The contents for each container element lists all or most of the possible returned response elements—both simple and subcontainer elements. Depending on your request, you may not see all elements in your response. If the value of an element is "empty," typically that element is omitted from the response.

Note: Elements annotated with an * (asterik) may appear multiple times in a response.

The second part of this Help topic is an element glossary for all single or "leaf" elements and subcontainer elements. Click here to go to the glossary or notice that each response element is a link—each response element is linked directly to its glossary entry.

access

Contains the set of effective permissions for a user or a group.

Contents:
- changePermissionsPrivilege
- executePrivilege
- modifyPrivilege
- readPrivilege

aclEntry

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

Contents:
- aclEntryId
- changePermissionsPrivilege
- executePrivilege
- modifyPrivilege
- readPrivilege
- principalName
- principalType

actualParameter

An actualParameter object provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from "formal parameters"—formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time.
Contents:
actualParameterId
actualParameterName
createTime
modifyTime
value

artifact
Contains elements to define the artifact. An artifact is specified by groupId and artifactKey. The name of an artifact is in this form "groupId:artifactKey". An artifact contains a collection of artifactVersions.

Contents:
artifactId
artifactKey
artifactName
artifactVersionNameTemplate
createTime
description
groupId
lastModifiedBy
modifyTime
owner
propertySheetId

artifactVersion
A "concrete" version of an artifact that contains a collection of files stored in the artifact repository.

Contents:
artifactKey
artifactName
artifactVersionId
artifactVersionName
artifactVersionState
buildNumber
createTime
majorMinorPatch
modifyTime
owner
propertySheetId
publisherJobId
publisherJobName
publisherJobStepId
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
**directoryProvider**

Contains information about the configuration used to communicate with an external directory service (LDAP or ActiveDirectory).

**Contents:**

- commonGroupNameAttribute
- createTime
- description
- directoryProviderId
- domaineName
- emailAttribute
- enableGroups
- fullUserNameAttribute
- groupBase
- groupMemberAttributes
- groupMemberFilter
- groupNameAttribute
- groupSearchFilter
- lastModifiedBy
- managerDn
- modifyTime
- name
- owner
- position
- propertySheetId
- providerIndex
- providerName
- providerType
- realm
- url
- useSSL
- userBase
- userNameAttribute
- userSearchFilter
- userSearchSubtree

**testDirectoryProvider**

Contains the results of testing a directory provider configuration as a list of test result blocks.

Each block contains a result with details about any failures. The **findGroupsTest** block also includes a list of groups for the test user.

The **findUserTest** block includes information about the user or users that matched the test user name.

**Contents:**

- findGroupsTest
  - testResult
  - details
groupList
group*
findUserTest
testResult
details
userList
userInfo*
email
fullUserName
mutable
providerName
userAuthenticationTest
testResult
details

**emailConfig**
Contains information about the configuration used to communicate with an email server.

Contents:
- **configName**
- **createTime**
- **description**
- **emailConfigId**
- **emailConfigName**
- **lastModifiedBy**
- **mailFrom**
- **mailHost**
- **mailPort**
- **mailProtocol**
- **mailUser**
- **modifyTime**
- **owner**
- **propertySheetId**

**emailNotifier**
Contains information about an email notifier.
Contents:
condition
cfgName
container
createTime
description
destinations
eEmailNotifierId
eventType
formattingTemplate
lastModifiedBy
modifyTime
notifierName
owner
propertySheetId

**formalParameter**

Contains information about a formal parameter.

Contents:
container
createTime
defaultValue
description
expansionDeferred
formalParameterId
formalParameterName
lastModifiedBy
modifyTime
owner
required
type

**gateway**

Contains information about a gateway.

Contents:
**group**

Contains information about a defined group of users.

Contents:
- createTime
- description
- gatewayDisabled
- gatewayId
- gatewayName
- hostName1
- hostName2
- lastModifiedBy
- modifyTime
- owner
- port1
- port2
- propertySheetId
- resourceName1
- resourceName2

**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:
jobStep

Contains information to define or locate a job step. Notice that the calledProcedure element (subcontainer element) can contain multiple jobStep elements.

Contents:

abortedBy
abortStatus
actualParameters*
callingState
combinedStatus
createTime
credentialName
deleted
directoryName
elapsedTime
errorCode
errorMessage
external
finish
jobId
jobName
jobStep*
lastModifiedBy
launchedByUser
licenseWaitTime
liveProcedure
liveSchedule
modifyTime
outcome
owner
priority
procedureName
projectName
propertySheet
propertySheetId
resourceWaitTime
runAsUser
scheduleName
start
status
steps
totalWaitTime
workspaceWaitTime

jobStep

Contains information to define or locate a job step. Notice that the calledProcedure element (subcontainer element) can contain multiple jobStep elements.

Contents:

abortedBy
abortStatus
actualParameters
alwaysRun
assignedResourceName
broadcast
calledProcedure
jobStep*
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
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
date
lastModifiedBy
modifyTime
owner
path
propertyId
propertyName
propertySheet*
propertySheetId
value

propertySheet

Contains one or more property elements.

Contents:
createTime
lastModifiedBy
modifyTime
owner
property*
propertySheetId
repository

Contains elements to define the artifact repository. The most useful elements in this object are "repositoryName" and "url". Clients publishing/retrieving artifact versions search repositories by name to obtain connection information.

Contents:

createTime
description
lastModifiedBy
modifyTime
owner
propertySheetId
repositoryDisabled
repositoryId
repositoryIndex
repositoryName
url
zoneName

resource

Contains elements to define a resource.

Contents:

agentState
alive
code
details
message
pingToken
protocolVersion
state
time
version
artifactCacheDirectory
createTime
description
lastRunTime
modifyTime
owner
pools
port
propertySheetId
proxyCustomization
proxyHostName
proxyPort
proxyProtocol
repositoryNames
resourceDisabled
resourceId
exclusiveJobId | 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
- attachedCredentials
- beginDate
- createTime
- credentialName
- description
- endDate
- interval
- intervalUnits
- lastModifiedBy
- lastRunTime
- misfirePolicy
- modifyTime
- owner
- priority
- procedureName
- projectName
- propertySheetId
- scheduleDisabled
- scheduleId
- scheduleName
- startTime
- stopTime
- timeZone
- weekDays

**serverStatus**

Contains elements to determine the status of the server.

Contents:

- apiMonitor
  - longestCall
api
callId
description
elapsedTime
label
remoteAddress
start
userName
mostActiveCalls
totalCallCount
activeCalls
call*
  api
callId
description
elapsedTime
label
remoteAddress
start
userName
recentCalls
call*
  api
callId
description
elapsedTime
label
remoteAddress
start
userName
lastMessage
messages
  message*
serverState
startTime
serverVersion

Contains elements to specify the ElectricFlow server version.

Contents:
- label
- protocolVersion
- schemaVersion
- version

state

Contains elements for a state in a running or completed workflow.

Contents:
- active
- createTime
- description
- errorMessage
- index
- lastModifiedBy
- modifyTime
- owner
- projectName
- propertySheetId
- stateId
- stateName
- subjob
- subprocedure
- subproject
- substartingState
- subworkflow
- subworkflowDefinition
- workflowName

stateDefinition

Contains elements for the state definition within a workflow definition.

Contents:
createTime
description
formalParameters
index
lastModifiedBy
modifyTime
owner
projectName
propertySheetId
startable
stateDefinitionId
stateDefinitionName
subprocedure
subproject
substartingState
subworkflowDefinition
workflowDefinitionName

**step**

Contains elements to specify or define a step.

Contents:

- actualParameters
- alwaysRun
- attachedCredentials
- attachedParameters
- broadcast
- command
- condition
- createTime
- credentialName*
- description
- errorHandling
- exclusive
- exclusiveMode
- postLogFileName
- postProcessor
- precondition
- procedureName
- projectName
- propertySheetId
- releaseExclusive
- releaseMode
- resourceName
- shell
- stepId
- stepName
- subprocedure
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
e-mail
fullUserName
groups
lastModifiedBy
modifyTime
mutable
owner
propertySheetId
providerName
userId
userName

**workflow**

Contains elements about a running or completed workflow.

Contents:
activeState
callingState
completed
createTime
ElectricFlow

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.

<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>abortStatus</td>
<td>enum</td>
<td>Possible values are: abort</td>
</tr>
<tr>
<td>abortedBy</td>
<td>string</td>
<td>The name of the user who aborted the job.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>aclEntryId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this aclEntry object.</td>
</tr>
<tr>
<td>active</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>activeCalls</td>
<td>subcontainer</td>
<td>A container element within the serverStatus element. activeCall describes an API currently running on the server.</td>
</tr>
<tr>
<td>activeState</td>
<td>string</td>
<td>The name of the activeState on the workflow object.</td>
</tr>
<tr>
<td>actualParameters</td>
<td>propertySheet</td>
<td>An actualParameter object provides the value for a parameter, which is passed to a procedure when it is invoked. Actual parameters can be provided for jobs and nested subprocedures within a job. Actual parameters are different from &quot;formal parameters&quot;- formal parameters define the parameters a procedure is expecting, and actual parameters provide values to use at run-time. For the workflow feature, these are the parameters that were passed when the workflow was started.</td>
</tr>
<tr>
<td>actualParameterId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this actual parameter object.</td>
</tr>
<tr>
<td>actualParameterName</td>
<td>string</td>
<td>The name of the parameter. This name is unique within the step, and at run time it matches the name of a formal parameter in the subprocedure.</td>
</tr>
<tr>
<td>admin</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>adminLicenseLastUse</td>
<td>date</td>
<td>The time at which the admin license was last used.</td>
</tr>
<tr>
<td>adminLicenseUser</td>
<td>string</td>
<td>The name of the user who is currently licensed as the &quot;admin&quot; user.</td>
</tr>
<tr>
<td>agentDrivePath</td>
<td>string</td>
<td>Drive-letter-based path used by Windows agents to access the workspace in steps.</td>
</tr>
<tr>
<td>agentUncPath</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>agentUnixPath</td>
<td>string</td>
<td>UNIX path used by UNIX agents and Linux ElectricFlow Web servers to access the workspace.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>agentState</td>
<td>subcontainer</td>
<td>A subcontainer element returned from certain resource queries. agentState returns specific information about an agent, including the state of the agent. Possible values are: unknown</td>
</tr>
<tr>
<td>alive</td>
<td>boolean</td>
<td>Refers to the agent state or status.</td>
</tr>
<tr>
<td>alwaysRun</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>api</td>
<td>string</td>
<td>An element returned on longestCall, activeCall, and recentCall subcontainers of the serverStatus element. api returns the API call (command) that is running or ran on the server.</td>
</tr>
<tr>
<td>apiMonitor</td>
<td></td>
<td>A server object that tracks API active, recent calls, and the total number of calls since server startup.</td>
</tr>
<tr>
<td>artifactCacheDirectory</td>
<td>string</td>
<td>The directory on the agent host where retrieved artifacts are stored.</td>
</tr>
<tr>
<td>artifactId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this artifact object.</td>
</tr>
<tr>
<td>artifactKey</td>
<td>string</td>
<td>User-specified identifier for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td>artifactName</td>
<td>string</td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td>artifactsDirectory</td>
<td>string</td>
<td>The directory in the workspace where you can put files to view, using a report link.</td>
</tr>
<tr>
<td>artifactVersionId</td>
<td>string</td>
<td>The unique ElectricFlow-generated ID for this artifact version object.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>name</td>
<td>The name of the artifact version. An artifact version name is interpreted by the server as the 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.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>artifactVersionNameTemplate</td>
<td>string</td>
<td>A template for the names of artifact versions published to this artifact. Over-rides the <code>global.artifactVersionNameTemplate</code>. The global setting can be manipulated in the Server Settings page (Administration &gt; Server, select the Settings link).</td>
</tr>
<tr>
<td>artifactVersionState</td>
<td>enum</td>
<td>Possible values are: <code>available</code></td>
</tr>
<tr>
<td>assignedResourceName</td>
<td>string</td>
<td>The name of the resource assigned to the step by the step scheduler.</td>
</tr>
<tr>
<td>attachedCredentials</td>
<td>list</td>
<td>The names of the credentials attached to the specified object.</td>
</tr>
<tr>
<td>attachedParameters</td>
<td>string</td>
<td>These are credential parameters that were attached to a step.</td>
</tr>
<tr>
<td>author</td>
<td>string</td>
<td>The author of the plugin.</td>
</tr>
<tr>
<td>autoDelete</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>batchRequests</td>
<td>string</td>
<td>A setting in the database configuration that determines whether or not to batch SQL queries when making a request to the database.</td>
</tr>
<tr>
<td>batchSize</td>
<td>string</td>
<td>The number of objects imported before being committed to the database.</td>
</tr>
<tr>
<td>beginDate</td>
<td>string</td>
<td><code>&lt;yyyy-mm-dd&gt;</code> The date the schedule is set to begin.</td>
</tr>
<tr>
<td>broadcast</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>buildNumber</td>
<td>string</td>
<td>User-defined build number component of the version attribute for the artifact version.</td>
</tr>
<tr>
<td>call</td>
<td>subcontainer</td>
<td>A subcontainer returned on <code>activeCall</code> and <code>recentCall</code> elements returned by the <code>serverStatus</code> API. <code>call</code> contains information specific to each API call on the server.</td>
</tr>
<tr>
<td>callId</td>
<td>number</td>
<td>A unique <code>ElectricFlow</code>-generated identifier for this particular call.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>callingState</td>
<td>string</td>
<td>The full property path to the &quot;calling state&quot;, which can appear on subjobs and subworkflows of a workflow.</td>
</tr>
<tr>
<td>calledProcedure</td>
<td>list</td>
<td>A subcontainer element within the jobStep element. The calledProcedure element can contain multiple jobStep elements.</td>
</tr>
<tr>
<td>category</td>
<td></td>
<td>(currently not used)</td>
</tr>
<tr>
<td>changePermissionsPrivilege</td>
<td>enum</td>
<td>Possible values are: allow</td>
</tr>
<tr>
<td>code</td>
<td>enum</td>
<td>Script to execute the functions for a step—passed to the step’s shell for execution.</td>
</tr>
<tr>
<td>combinedStatus</td>
<td>enum</td>
<td>More inclusive step status output - this value may combine up to three sub-elements: status</td>
</tr>
<tr>
<td>command</td>
<td>string</td>
<td>The command to run steps - for command steps.</td>
</tr>
<tr>
<td>commonGroupNameAttribute</td>
<td>string</td>
<td>The attribute in a group record that contains the common group name. If specified, this name is used only when searching for groups from an external provider.</td>
</tr>
<tr>
<td>completed</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>completeUserName</td>
<td>string</td>
<td>A SQL server-specific tag that includes the user's name and the user's domain name.</td>
</tr>
<tr>
<td>concurrentResources</td>
<td>object</td>
<td>A subcontainer element that includes information about &quot;in use&quot; and &quot;maximum licensed&quot; hosts and proxied hosts for the licenseUsage API command.</td>
</tr>
<tr>
<td>concurrentSteps</td>
<td>number</td>
<td>The total number of steps running at the same time in the ElectricFlow system. This means all steps from all procedures, regardless of how many or how few projects you have created.)</td>
</tr>
<tr>
<td>concurrentUsers</td>
<td>object</td>
<td>A subcontainer element that includes information about the admin license, &quot;in use&quot; licenses, and the maximum number of licenses for the licenseUsage API command.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td>-------------------</td>
</tr>
</tbody>
</table>
| condition        | string  | *For steps:*  
If empty or non-zero, the step will run. If set to "0", the step is skipped. A useful setting during procedure development or when re-running a job that has already completed some of the steps. Also, this argument is useful for conditional execution of steps based on properties set by earlier steps.  
*For email notifiers:*  
Mail sent if the condition evaluates to "true". The condition is a string subject to property expansion. The notification will NOT be sent if the expanded string is "false" or "0". If no condition is specified, the notification is ALWAYS sent. |
| configName       | string  | The name of the configuration. |
| container        | string  | An object ID for a "container" that contains formal parameters.  
In another context, this is typically the type and name of the workflow or job with a corresponding ID. |
| containerName    | string  | The name of the container. |
| createTime       | date    | The time when this object was created. |
| credentialId     | number  | The unique ElectricFlow-generated ID for this credential object. |
| credentialName   | string  | credentialName can be one of two forms:  
relative (for example, "cred1") - the credential is assumed to be in the project that contains the request target object. Requires a qualifying project name.  
absolute (for example, "/projects/BuildProject/credentials/cred1") - the credential can be from any specified project, regardless of the target object’s project. |
<p>| customDatabaseDialect | string | Class name for the Hibernate dialect. The server chooses an appropriate dialect based on databaseType or this can be part of the custom specification. |
| customDatabaseDriver | string | Class name of the JDBC driver. The server will choose an appropriate driver based on databaseType or this can be part of the custom specification. |
| customDatabaseUrl | string  | The JDBC URL to use. The server will compose an appropriate URL or this can be part of the custom specification. |</p>
<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>customerName</td>
<td>string</td>
<td>The name of a company and/or group name with a company that is using ElectricFlow.</td>
</tr>
<tr>
<td>databaseDialect</td>
<td>string</td>
<td>Class name for the Hibernate dialect (the server chooses an appropriate dialect based on databaseType).</td>
</tr>
<tr>
<td>databaseDriver</td>
<td>string</td>
<td>Class name of the JDBC driver (the server will choose an appropriate driver based on databaseType).</td>
</tr>
<tr>
<td>databaseName</td>
<td>string</td>
<td>The name of the database the ElectricFlow server is using.</td>
</tr>
<tr>
<td>databaseType</td>
<td>enum</td>
<td>Possible values are: builtin</td>
</tr>
<tr>
<td>databaseUrl</td>
<td>string</td>
<td>The JDBC URL to use (the server will compose an appropriate URL).</td>
</tr>
<tr>
<td>defaultValue</td>
<td>string</td>
<td>This value is used for the formal parameter if a value is not supplied by the caller.</td>
</tr>
<tr>
<td>delayUntil</td>
<td>date</td>
<td>For a step that was rescheduled due to a resource or workspace problem, this is the next time when the step will be eligible to run.</td>
</tr>
<tr>
<td>deleted</td>
<td>byte</td>
<td>The object was marked for background deletion. Possible values are &quot;0&quot; or &quot;1&quot;. Default is &quot;0&quot; (not set).</td>
</tr>
<tr>
<td>dependentArtifacts</td>
<td>string</td>
<td>A space-separated list of artifacts.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td>A plain text or HTML description for this object. If using HTML, you must surround your text with &lt;html&gt; ... &lt;/html&gt; tags. The only HTML tags allowed in the text are: &lt;a&gt; &lt;b&gt; &lt;br&gt; &lt;div&gt; &lt;dl&gt; &lt;font&gt; &lt;i&gt; &lt;il&gt; &lt;ol&gt; &lt;p&gt; &lt;pre&gt; &lt;span&gt; &lt;style&gt; &lt;table&gt; &lt;tc&gt; &lt;td&gt; &lt;th&gt; &lt;tr&gt; &lt;ul&gt;</td>
</tr>
<tr>
<td>destinations</td>
<td>string</td>
<td>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.</td>
</tr>
<tr>
<td>details</td>
<td>string</td>
<td>A string containing details about agent status.</td>
</tr>
<tr>
<td>directoryName</td>
<td>string</td>
<td>The name of the job's directory within each workspace for a job.</td>
</tr>
<tr>
<td>directoryProviderId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this directory provider object.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>domainName</td>
<td>string</td>
<td>The name of the domain from which the Active Directory servers are automatically discovered.</td>
</tr>
<tr>
<td>elapsedTime</td>
<td>number</td>
<td>The number of milliseconds between the start and end times for the job or job step - or a workflow.</td>
</tr>
<tr>
<td>email</td>
<td>string</td>
<td>The user's email address.</td>
</tr>
<tr>
<td>emailAttribute</td>
<td>string</td>
<td>The attribute in a user record that contains the user's email address. If the attribute is not specified, the account name and domain name are concatenated to form an email address.</td>
</tr>
<tr>
<td>emailConfigId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this email configuration object.</td>
</tr>
<tr>
<td>emailConfigName</td>
<td>string</td>
<td>The name of the email configuration.</td>
</tr>
<tr>
<td>emailNotifierId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this email notifier object.</td>
</tr>
<tr>
<td>enableGroups</td>
<td>boolean</td>
<td>Determines whether or not to enable external groups for the directory provider.</td>
</tr>
<tr>
<td>endDate</td>
<td>string</td>
<td><code>&lt;yyyymm-dd&gt;</code> The date this schedule is set to end.</td>
</tr>
<tr>
<td>errorCode</td>
<td>enum</td>
<td>Displays the error code, identifying which error occurred.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>errorHandling</td>
<td>enum</td>
<td>Determines what happens to the procedure if the step fails:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- failProcedure - The current procedure continues, but the overall status is error (default).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- abortProcedure - Aborts the current procedure, but allows already-running steps in the current procedure to complete.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- abortProcedureNow - Aborts the current procedure and terminates running steps in the current procedure.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- abortJob - Aborts the entire job, terminates running steps, but allows alwaysRun steps to run.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- abortJobNow - Aborts the entire job and terminates all running steps, including alwaysRun steps.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ignore - Continues as if the step succeeded.</td>
</tr>
<tr>
<td>errorMessage</td>
<td>string</td>
<td>A description of the error.</td>
</tr>
<tr>
<td>evaluation</td>
<td>boolean</td>
<td>Determines whether or not this license is an evaluation copy only.</td>
</tr>
<tr>
<td>eventType</td>
<td>enum</td>
<td>Possible values are: onCompletion</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&quot;onStart&quot; triggers an event when the job or job step begins. &quot;onCompletion&quot; triggers an event when the job finishes, no matter how it finishes. Default is &quot;onCompletion&quot;.</td>
</tr>
<tr>
<td>exclusive</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>exclusiveJobId</td>
<td>number</td>
<td>The ID number of the job that owns this resource, which occurs when one of the job’s steps requests exclusive use of the resource for the duration of the job.</td>
</tr>
<tr>
<td>exclusiveJobName</td>
<td>string</td>
<td>The name of the job that owns this resource, which occurs when one of the job’s steps requests exclusive use of the resource for the duration of the job.</td>
</tr>
<tr>
<td>exclusiveJobStepId</td>
<td>number</td>
<td>The ID number of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>exclusiveJobStepName</td>
<td>name</td>
<td>The name of the job step that owns this resource, which occurs when one of the steps request exclusive use of the resource for the duration of the job.</td>
</tr>
<tr>
<td>exclusiveMode</td>
<td>enum</td>
<td>Possible values are: none</td>
</tr>
<tr>
<td>executePrivilege</td>
<td>enum</td>
<td>Possible values are: allow</td>
</tr>
<tr>
<td>exitCode</td>
<td>number</td>
<td>The step's exit code.</td>
</tr>
<tr>
<td>expandable</td>
<td>boolean</td>
<td>&lt;Boolean flag -0</td>
</tr>
<tr>
<td>expansionDeferred</td>
<td>boolean</td>
<td>&lt;Boolean flag -0</td>
</tr>
<tr>
<td>expiration</td>
<td>date</td>
<td>The date when a user license expires.</td>
</tr>
<tr>
<td>expirationDate</td>
<td>date</td>
<td>The date when a license expires.</td>
</tr>
<tr>
<td>external</td>
<td>boolean</td>
<td>&lt;Boolean flag -0</td>
</tr>
<tr>
<td>featureName</td>
<td>string</td>
<td>The name of the licensed feature. Possible features include: Server</td>
</tr>
<tr>
<td>findGroupsTest</td>
<td>subcontainer</td>
<td>For the testDirectoryProvider API, this element provides information on which groups the user is a member.</td>
</tr>
<tr>
<td>findUserTest</td>
<td>subcontainer</td>
<td>For the testDirectoryProvider API, this element contains specific information about the user.</td>
</tr>
<tr>
<td>finish</td>
<td>date</td>
<td>The time the job or workflow completed.</td>
</tr>
<tr>
<td>formalParameterId</td>
<td>number</td>
<td>The formal parameter's ID.</td>
</tr>
<tr>
<td>formalParameterName</td>
<td>string</td>
<td>The name of the procedure's parameter, containing a credential reference.</td>
</tr>
<tr>
<td>formalParameters</td>
<td>string</td>
<td>The parameters that must be supplied when entering the state (similar to formal parameters on a procedure).</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>formattingTemplate</td>
<td>string</td>
<td>Specifies a template for formatting email messages when an event [notification] is triggered by the emailNotifier.</td>
</tr>
<tr>
<td>fullUserName</td>
<td>string</td>
<td>The user's full name - not his or her nickname.</td>
</tr>
<tr>
<td>fullUserNameAttribute</td>
<td>string</td>
<td>The attribute in a user record that contains the user's full name (first and last) for display in the UI. If this attribute is not specified or the resulting value is empty, the user's account name is used instead.</td>
</tr>
<tr>
<td>gatewayDisabled</td>
<td>boolean</td>
<td>&lt;Boolean flag -0</td>
</tr>
<tr>
<td>gatewayId</td>
<td>number</td>
<td>The ElectricFlow-generated ID number for this gateway.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>string</td>
<td>The name of the gateway.</td>
</tr>
<tr>
<td>gateways</td>
<td>list</td>
<td>A space-separated list of gateway names.</td>
</tr>
<tr>
<td>gracePeriod</td>
<td>number</td>
<td>The number of days available after the ElectricFlow license expires.</td>
</tr>
<tr>
<td>groupBase</td>
<td>string</td>
<td>This string is prepended to the basedn to construct the directory DN that contains group records.</td>
</tr>
<tr>
<td>groupId</td>
<td>number</td>
<td>The unique ElectricFlow-generated group ID.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>For Artifact Management:</strong> A user-generated group name for this artifact. This field is limited to alphanumeric characters, spaces, underscores, hyphens, and periods.</td>
</tr>
<tr>
<td>groupList</td>
<td>list</td>
<td>For the testDirectoryProvider API, this element contains zero or more groups returned after querying existing groups known to the directory provider.</td>
</tr>
<tr>
<td>groupMemberAttributes</td>
<td>string</td>
<td>A comma-separated attribute name list that identifies a group member. Most LDAP configurations only specify a single value, but if there is a mixture of POSIX and LDAP style groups in the directory, multiple attributes might be required.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>groupMemberFilter</td>
<td>string</td>
<td>This LDAP query is performed in the groups directory context to identify groups containing a specific user as a member. Two common forms of group record in LDAP directories: POSIX style groups where members are identified by account name, and uniqueGroupOfNames records where members are identified by the full user DN. Both forms are supported, so the query is passed to parameters: &quot; (0)&quot; is replaced with the full user record DN, and &quot; (1) &quot; is replaced with the user's account name.</td>
</tr>
<tr>
<td>groupName</td>
<td>string</td>
<td>The full name of a group. For Active Directory and LDAP, this is a full DN.</td>
</tr>
<tr>
<td>groupNameAttribute</td>
<td>string</td>
<td>The group record attribute that contains the name of the group.</td>
</tr>
<tr>
<td>groups</td>
<td>list</td>
<td>A space-separated list of group names.</td>
</tr>
<tr>
<td>groupSearchFilter</td>
<td>string</td>
<td>The LDAP query performed in the context of the groups directory to enumerate group records.</td>
</tr>
<tr>
<td>groupSettingsId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this group settings object.</td>
</tr>
<tr>
<td>hostName</td>
<td>string</td>
<td>The computer name or IP address for the machine containing the ElectricFlow server or agent.</td>
</tr>
<tr>
<td>hostName1</td>
<td>string</td>
<td>For gateways: The name Resource 2 uses to communicate with Resource 1. If &quot;blank&quot;, the Agent Host Name attribute in Resource 1's definition is used at runtime.</td>
</tr>
<tr>
<td>hostName2</td>
<td>string</td>
<td>For gateways: The name Resource 1 uses to communicate with Resource 2. If &quot;blank&quot;, the Agent Host Name attribute in Resource 2's definition is used at runtime.</td>
</tr>
<tr>
<td>hostOS</td>
<td>string</td>
<td>The full name of the host operating system, plus its version. However, if this host is a proxy, the value is &quot;proxied&quot;.</td>
</tr>
<tr>
<td>hostPlatform</td>
<td>string</td>
<td>Examples for &quot;platform&quot; are: Windows, Linux, HPUX, and so on. However, if this host is a proxy, the value is &quot;proxied&quot;.</td>
</tr>
<tr>
<td>index</td>
<td>number</td>
<td>The numeric index of the transition that indicates its order in the list of transitions in a state definition.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>-------------------</td>
</tr>
<tr>
<td>interval</td>
<td>string</td>
<td>The repeat interval for starting new jobs.</td>
</tr>
<tr>
<td>intervalUnits</td>
<td>enum</td>
<td>Possible values are: hours</td>
</tr>
<tr>
<td>inUseHosts</td>
<td>number</td>
<td>The number of hosts (agents) currently in use.</td>
</tr>
<tr>
<td>inUseLicenses</td>
<td>number</td>
<td>The number of user licenses currently in use.</td>
</tr>
<tr>
<td>inUseProxiedHosts</td>
<td>number</td>
<td>The number of proxy target hosts currently in use.</td>
</tr>
<tr>
<td>jobId</td>
<td>number</td>
<td>The unique ElectricFlow-generated identifier (a UUID) for a job that is assigned automatically when the job is created. The system also accepts a job name assigned to the job by its name template.</td>
</tr>
<tr>
<td>jobName</td>
<td>string</td>
<td>The name of the job.</td>
</tr>
<tr>
<td>jobNameTemplate</td>
<td>string</td>
<td>Template used to determine the default name of jobs launched from a procedure.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>number</td>
<td>The unique identifier for a job step, assigned automatically when the job step is created.</td>
</tr>
<tr>
<td>jobStepName</td>
<td>string</td>
<td>The name of the job step.</td>
</tr>
<tr>
<td>label</td>
<td>string</td>
<td>A name used by a plugin for display in a list, or this may represent context-specific info about an API call—not all API calls return a &quot;label&quot; tag.</td>
</tr>
<tr>
<td>lastMessage</td>
<td>string</td>
<td>Element returned by the serverStatus API showing the last message the server received.</td>
</tr>
<tr>
<td>lastModifiedBy</td>
<td>string</td>
<td>Shows who (generally a user name) last modified the object.</td>
</tr>
<tr>
<td>lastResourceUsed</td>
<td>string</td>
<td>The name of the most recently used resource from the pool.</td>
</tr>
<tr>
<td>lastRunTime</td>
<td>date</td>
<td>The last time a job was launched by a schedule. -or- In a resource response, this is the most recent time that a job step ran on the resource.</td>
</tr>
<tr>
<td>lastUse</td>
<td></td>
<td>Returned element in the concurrentUsers subcontainer (for the licenseUsage API), providing the last time a specific user accessed ElectricFlow.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>launchedByUser</td>
<td>string</td>
<td>The name of the user or project principal that explicitly launched the job. This property is blank when the job is launched by a schedule.</td>
</tr>
<tr>
<td>licenseId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this license.</td>
</tr>
<tr>
<td>licenseWaitTime</td>
<td></td>
<td>The amount of time a job step was stalled waiting for an available license. On a job, this is the sum of license wait for all job steps.</td>
</tr>
<tr>
<td>liveProcedure</td>
<td>string</td>
<td>Shows the current procedure name for the procedure step from which the job or job step was created – if the procedure step was renamed since the job or job step was launched, this is the procedure step’s new name, and if the procedure step was deleted, this will be null.</td>
</tr>
<tr>
<td>liveProcedureStep</td>
<td>string</td>
<td>Shows the current procedure step name for the procedure step from which the job step was created – if the procedure step was renamed since the job was launched, this is the procedure step’s new name, and if the procedure step was deleted, this will be null.</td>
</tr>
<tr>
<td>liveSchedule</td>
<td>string</td>
<td>Shows the current schedule name for the procedure step from which the job was created – if the schedule was renamed since the job was launched, this is the schedule’s new name, and if the schedule was deleted, this will be null.</td>
</tr>
<tr>
<td>liveWorkflowDefinition</td>
<td>string</td>
<td>Shows the current workflow definition name for the workflow definition from which the workflow was created – if the workflow definition was renamed since the workflow was launched, this is the workflow definition’s new name, and if the workflow definition was deleted, this will be null.</td>
</tr>
<tr>
<td>local</td>
<td>boolean</td>
<td>`&lt;Boolean flag&gt; 0</td>
</tr>
<tr>
<td>logEntryId</td>
<td>number</td>
<td>The ElectricFlow-generated ID number for the log entry record.</td>
</tr>
<tr>
<td>logFileName</td>
<td>string</td>
<td>A custom log file name produced by running the step. By default, ElectricFlow assigns a unique name for this file.</td>
</tr>
<tr>
<td>longestCall</td>
<td>string</td>
<td>Provides the API call that took the longest time.</td>
</tr>
<tr>
<td>mailFrom</td>
<td>string</td>
<td>The email address used as the email sender address for notifications.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mailHost</td>
<td>string</td>
<td>The name of the email server host.</td>
</tr>
<tr>
<td>mailPort</td>
<td>number</td>
<td>The port number for the mail server, but may not need to be specified. The protocol software determines the default value (25 for SMTP and 465 for SSMTP). Specify a value for this argument when a non-default port is used.</td>
</tr>
<tr>
<td>mailProtocol</td>
<td>string</td>
<td>This is either SSMTP or SMTP (not case-sensitive). The default is SMTP.</td>
</tr>
<tr>
<td>mailUser</td>
<td>string</td>
<td>This can be an individual or a generic name like &quot;ElectricFlow&quot; - name of the email user on whose behalf ElectricFlow sends email notifications.</td>
</tr>
<tr>
<td>majorMinorPatch</td>
<td>string</td>
<td>major.minor.patch component of the version attribute for the artifact.</td>
</tr>
<tr>
<td>managerDn</td>
<td>string</td>
<td>The name of a user who has read-only access to the LDAP or Active Directory server. Typically a DN (distinguished name). A simple name may be used when the Active Directory server's URL is being auto-discovered via DNS. <strong>Note:</strong> This user does not need to be an admin user with modify privileges.</td>
</tr>
<tr>
<td>maxConcurrentSteps</td>
<td>number</td>
<td>The maximum number of steps that can run at the same time per the provisions of your ElectricFlow license.</td>
</tr>
<tr>
<td>maxHosts</td>
<td>number</td>
<td>The maximum number of hosts licensed for resource use.</td>
</tr>
<tr>
<td>maxLicenses</td>
<td>number</td>
<td>The maximum number of licenses available for users.</td>
</tr>
<tr>
<td>maxProxiedHosts</td>
<td>number</td>
<td>The maximum number of available licenses for proxy hosts.</td>
</tr>
<tr>
<td>message</td>
<td>string</td>
<td>A user-readable diagnostic message associated with an error.</td>
</tr>
<tr>
<td>messages</td>
<td>list</td>
<td>Multiple error or diagnostic messages.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| misfirePolicy      | enum   | Possible values are: ignore | run once  
A schedule may not fire at the allotted time because a prior job is still running, the server is running low on resources and there is a delay, or the server is down. When the underlying issue is resolved, the server will schedule the next job at the next regularly scheduled time slot if the policy is 'ignore', otherwise it will run the job immediately. Defaults to "ignore". |
| modifyPrivilege    | enum   | Possible values are: allow | deny | inherit |
| modifyTime         | date   | The time when the object was last modified.                                                                                                   |
| monthDays          | string | Restricts the schedule to specified days of the month. Specify numbers from 1-31, separating multiple numbers with a space.                     |
| mostActiveCalls    | number | The number of most active API calls since server startup.                                                                                      |
| mutable            | boolean| If "true," the member list of this group is editable within ElectricFlow via the web UI or the modifyGroup API.                                  |
| name               | string | The name of the directory provider.                                                                                                           |
| notifierName       | string | The name of the email notifier.                                                                                                                |
| objectId           | number | An object identifier returned by findObjects and getObjects. This value is a "handle" only for passing to API commands. The internal structure of this value is subject to change - do not parse this value. |
| objectName         | string | The name of the object.                                                                                                                       |
| objectType         | enum   | The type of object being described, for example: project, procedure, step, and so on.                                                       |
| orderingFilter     | string | A Javascript block invoked when scheduling resources for a pool.  
**Note:** A Javascript block is not required unless you need to override the default resource ordering behavior. |
<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
</table>
| outcome          | enum  | Possible values for outcome:
|                  |       | Note: The outcome is accurate only if the job status is
|                  |       | "completed."
|                  |       | success - The job finished successfully.
|                  |       | warning - The job completed with no errors, but
|                  |       | encountered some suspicious conditions.
|                  |       | error - The job has finished execution with errors. |
| owner            | string| The person (user name) who created the object. |
| parallel         | boolean| `<Boolean flag - 0|1|true|false>` - If set, indicates
|                  |       | this step should run at the same time as adjacent
|                  |       | steps marked to run as parallel also. Defaults to
|                  |       | "false". |
| password         | string| The password matching the specified user name. |
| path             | string| The property path that specifies the object to use. |
| pingToken        | number| Every time an agent starts, a unique pingToken value
|                  |       | is generated. The server uses the pingToken value to
|                  |       | determine agent restarts by noticing the values before
|                  |       | and after a restart. |
| pluginId         | number| The unique ElectricFlow-generated ID for the plugin
|                  |       | object. |
| pluginKey        | string| The name of the plugin as displayed on the
|                  |       | ElectricFlow Plugin Manager web page. |
| pluginName       | string| The name of the plugin - the plugin key for a promoted
|                  |       | plugin or a plugin key and version for an unpromoted
|                  |       | plugin. |
| pluginVersion    | string| The version of the plugin being described. |
| pools            | list  | A space-separated list of one or more pool names
|                  |       | where this resource is a member. Steps defined to run
|                  |       | on a resource pool will run on any available member
|                  |       | (resource) in the pool. |
| port             | number| If a port number is not specified, the default
|                  |       | ElectricFlow port is used.
|                  |       | For a proxy resource, this is the port number for the
|                  |       | service running on the proxy target that will run
|                  |       | commands on behalf of the ElectricFlow agent. For
|                  |       | ssh, the default is 22. |
| port1            | number| The port number used by Gateway Resource1 -
<p>|                  |       | default is to the port number used by the resource. |</p>
<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>port2</td>
<td>number</td>
<td>The port number used by Gateway Resource2 - default is to the port number used by the resource.</td>
</tr>
<tr>
<td>position</td>
<td>number</td>
<td>Used to reorder an ElectricFlow object. For example, if reordering directory providers: the provider name is moved to a position just before this provider. &quot;Blank&quot; means move the provider to the end of the provider list.</td>
</tr>
<tr>
<td>postExitCode</td>
<td>number</td>
<td>The step's post processor exit code.</td>
</tr>
<tr>
<td>postLogFileName</td>
<td>string</td>
<td>The log file name produced by this step's post processor.</td>
</tr>
<tr>
<td>postProcessor</td>
<td>string</td>
<td>This program looks at the step output to find errors and warnings. ElectricFlow includes a customizable program called &quot;postp&quot; for this purpose. The value for postProcessor is a command string for invoking a post-processor program in the platform shell for the resource (cmd for Windows, sh for UNIX).</td>
</tr>
<tr>
<td>precondition</td>
<td>string</td>
<td>Set this property to make a step wait until one or more dependent conditions are met. When a job step is eligible to transition from pending to runnable, a precondition is evaluated. A precondition is a fixed text or text embedding property reference that is evaluated to TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE. The step will block until the precondition is TRUE.</td>
</tr>
<tr>
<td>principal</td>
<td>string</td>
<td>The user or project principal from the session that was active when the event occurred.</td>
</tr>
<tr>
<td>principalName</td>
<td>string</td>
<td>This is either a user or a group name.</td>
</tr>
<tr>
<td>principalType</td>
<td>enum</td>
<td>Possible values are: group</td>
</tr>
</tbody>
</table>
| priority         | enum      | Possible values are: low|normal|high|highest
Priorities take effect when two or more job steps in different jobs are waiting for the same resource. When the resource is available, it will be used by the job step that belongs to the job with the highest priority. If the priority level is the same, the resource will be used by the job step that belongs to the job with the lowest job ID number. If the job steps are in the same job, the resource will be used first by the step with the lowest job step ID number. |
<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureId</td>
<td>number</td>
<td>The unique ElectricFlow-generated procedure ID.</td>
</tr>
<tr>
<td>procedureName</td>
<td>string</td>
<td>The name of the procedure - may be a path to the procedure.</td>
</tr>
<tr>
<td>productName</td>
<td>string</td>
<td>The name of the product with the licensed feature. Possible products include: ElectricFlow</td>
</tr>
<tr>
<td>project</td>
<td>name</td>
<td>The name of the project associated with the plugin.</td>
</tr>
<tr>
<td>projectId</td>
<td>number</td>
<td>The unique ElectricFlow-generated project ID.</td>
</tr>
<tr>
<td>projectName</td>
<td>string</td>
<td>The name of the project - may be a path. The project name is ignored for credentials, procedure, steps, and schedules if it is specified as a path.</td>
</tr>
<tr>
<td>promoted</td>
<td>boolean</td>
<td>`&lt;Boolean flag · 0</td>
</tr>
<tr>
<td>propertyId</td>
<td>number</td>
<td>The unique ElectricFlow-generated property ID.</td>
</tr>
<tr>
<td>propertyName</td>
<td>string</td>
<td>The name of the property. It may be a relative or absolute property path, including &quot;my&quot; paths such as &quot;/myProject/prop1&quot;.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>number</td>
<td>The unique identifier for a property sheet, assigned automatically when the property sheet is created.</td>
</tr>
<tr>
<td>protocolVersion</td>
<td>string</td>
<td>The server API protocol version. For example, the server accepts messages from ectool and ec-perl.</td>
</tr>
<tr>
<td>providerIndex</td>
<td>number</td>
<td>The index that specifies the search order across multiple directory providers. For example: 2 LDAP providers, one with index &quot;0&quot; and one with index &quot;1&quot; means the providers will be searched in that numerical order.</td>
</tr>
<tr>
<td>providerName</td>
<td>string</td>
<td>The LDAP or Active Directory provider name.</td>
</tr>
<tr>
<td>providerType</td>
<td>enum</td>
<td>Possible values are: ldap</td>
</tr>
<tr>
<td>proxyCustomization</td>
<td>string</td>
<td>Perl code customizing how the proxy resource communicates with the proxy target. This argument is applicable only for proxy resources.</td>
</tr>
<tr>
<td>proxyHostName</td>
<td>string</td>
<td>The name or IP address of the computer containing the ElectricFlow Agent used for a proxy resource.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>proxyPort</td>
<td>number</td>
<td>The ElectricFlow agent port number for a proxy resource.</td>
</tr>
<tr>
<td>proxyProtocol</td>
<td>string</td>
<td>Protocol for communicating with the proxy target. Defaults to ssh. (This argument is not exposed in the ElectricFlow Web Interface at this time.)</td>
</tr>
<tr>
<td>publisherJobId</td>
<td>number</td>
<td>The ElectricFlow-generated ID for the job that published the artifact version.</td>
</tr>
<tr>
<td>publisherJobName</td>
<td>name</td>
<td>The name of the job that published the artifact version.</td>
</tr>
<tr>
<td>publisherJobStepId</td>
<td>number</td>
<td>The ElectricFlow-generated ID for the job step that published the artifact version.</td>
</tr>
<tr>
<td>qualifier</td>
<td>string</td>
<td>User-defined qualifier component of the version attribute for the artifact.</td>
</tr>
<tr>
<td>readPrivilege</td>
<td>enum</td>
<td>Possible values are: allow</td>
</tr>
<tr>
<td>realm</td>
<td>string</td>
<td>The realm of the LDAP directory provider—used to create unique user names when there are multiple providers.</td>
</tr>
<tr>
<td>recentCall</td>
<td>subcontainer</td>
<td>A subcontainer element on the serverStatus API - a call no longer active (completed). The API monitor keeps track of the 10 most recent calls.</td>
</tr>
<tr>
<td>releaseExclusive</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>releaseMode</td>
<td>string</td>
<td>Possible values are: none</td>
</tr>
<tr>
<td>remoteAddress</td>
<td>string</td>
<td>Generally a combined IP address plus a port specification - used when the agent is talking to the server or to show where the request to the server originated.</td>
</tr>
<tr>
<td>repositoryDisabled</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>repositoryId</td>
<td>number</td>
<td>The ElectricFlow-generated ID for the artifact repository.</td>
</tr>
<tr>
<td>repositoryIndex</td>
<td>integer</td>
<td>The order of the repository within a list of repositories.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>string</td>
<td>The name of the artifact repository.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>repositoryNames</td>
<td>list</td>
<td>A list of one or more repository server names—each repository name listed on a &quot;new line&quot;.</td>
</tr>
<tr>
<td>required</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>resourceDisabled</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>resourceId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this resource.</td>
</tr>
<tr>
<td>resourceName1</td>
<td>string</td>
<td>The name for the first of two resources required to create a gateway. &quot;Spaces&quot; are NOT allowed in a resource name.</td>
</tr>
<tr>
<td>resourceName2</td>
<td>string</td>
<td>The name for the second of two resources required to create a gateway. &quot;Spaces&quot; are NOT allowed in a resource name.</td>
</tr>
<tr>
<td>resourceName</td>
<td>string</td>
<td>The name of a resource.</td>
</tr>
<tr>
<td>resourceNames</td>
<td>string</td>
<td>A list of strings that refer to resources that belong to the pool. Names that do not refer to existing resources are ignored.</td>
</tr>
<tr>
<td>resourcePoolDisabled</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>resourcePoolId</td>
<td>number</td>
<td>The unique ID number for a resource pool.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>name</td>
<td>The name of the resource pool.</td>
</tr>
<tr>
<td>resources</td>
<td>string</td>
<td>A space-separated list of resource names.</td>
</tr>
<tr>
<td>resourceUsageId</td>
<td>number</td>
<td>The unique ID number of the resource usage record.</td>
</tr>
<tr>
<td>resourceWaitTime</td>
<td></td>
<td>The amount of time a job step waited for a resource to become available. On a job, this is the sum of time all job steps waited for resource availability. This could indicate that eligible resources for the step have reached their step limit, are in-use but the step requires a resource exclusively, or resources are down.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>------------------</td>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>retries</td>
<td>number</td>
<td>The number of attempts to write to the step log in the workspace. In a running step, this is the number of retries attempted up to this point. The most common reason for step retries is the workspace for the step was unavailable.</td>
</tr>
<tr>
<td>retrievers</td>
<td>list</td>
<td>A collection of retrieve elements that can contain a jobName, jobId, and/or a jobStepId elements.</td>
</tr>
<tr>
<td>runAsUser</td>
<td>string</td>
<td>The name of the user being impersonated in this job.</td>
</tr>
<tr>
<td>runnable</td>
<td>date</td>
<td>The time when the step became runnable.</td>
</tr>
<tr>
<td>runningSteps</td>
<td></td>
<td>The number of steps running at the same time.</td>
</tr>
<tr>
<td>runtime</td>
<td>number</td>
<td>The number of milliseconds the step command spent running on a resource.</td>
</tr>
<tr>
<td>scheduleDisabled</td>
<td>boolean</td>
<td>`&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>scheduleId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for the schedule.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>string</td>
<td>The name of the schedule - may be a path to the schedule.</td>
</tr>
<tr>
<td>schemaVersion</td>
<td>number</td>
<td>The ElectricFlow server's database schema version.</td>
</tr>
<tr>
<td>serverState</td>
<td>enum</td>
<td>Possible values are: bootstrap, databaseConfiguration, databaseConnection, databaseSchema, running, failed, stopping, importFailed</td>
</tr>
<tr>
<td>severity</td>
<td>enum</td>
<td>Possible values are: INFO</td>
</tr>
<tr>
<td>shell</td>
<td>string</td>
<td>Where shell is the name of a program used to execute commands contained in the &quot;command&quot; field. Normally, this file is a command shell, but it could be any other command line program. The default is &quot;cmd /q /c&quot; for a Windows agent and &quot;sh -e&quot; for a UNIX agent. This is applicable to command steps only.</td>
</tr>
<tr>
<td>signature</td>
<td>string</td>
<td>The digital signature on this license.</td>
</tr>
<tr>
<td>start</td>
<td>date</td>
<td>The time this job or workflow began executing.</td>
</tr>
<tr>
<td>startable</td>
<td>boolean</td>
<td>&quot;True&quot; means this state definition can be the initial state of an instantiated workflow.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>startingState</td>
<td>string</td>
<td>The initial state of the workflow.</td>
</tr>
<tr>
<td>startTime</td>
<td>string</td>
<td>Formatted hh:mm, using the 24-hour clock. Using this schedule, ElectricFlow starts creating jobs at this time on the specified days.</td>
</tr>
<tr>
<td>stateDefinitionId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this state definition object.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>string</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this state object.</td>
</tr>
<tr>
<td>statementCacheSize</td>
<td>string</td>
<td>The number of MS SQL statements cached in the database.</td>
</tr>
<tr>
<td>stateName</td>
<td>string</td>
<td>The name of the state.</td>
</tr>
<tr>
<td>status</td>
<td>enum</td>
<td>Possible values for status: pending - The job is not yet runnable—it is waiting for other steps to complete first.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>runnable - The job is ready to run, but it is waiting for a resource to become available.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>running - The job is assigned to a resource and is executing the step command.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>completed - The job finished executing.</td>
</tr>
<tr>
<td>stepCount</td>
<td>number</td>
<td>The number of executing steps on this resource.</td>
</tr>
<tr>
<td>stepErrorCode</td>
<td>enum</td>
<td>Agent error messages.</td>
</tr>
<tr>
<td>stepId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for the step.</td>
</tr>
<tr>
<td>stepLimit</td>
<td>number</td>
<td>The number of steps that can run on the resource at one time. (Previously setting the limit to 1 enforces serial access to the resource.)</td>
</tr>
<tr>
<td>stepName</td>
<td>string</td>
<td>The name of the step - may be a path to the step.</td>
</tr>
<tr>
<td>steps</td>
<td></td>
<td>The list or number of steps in a job.</td>
</tr>
<tr>
<td>stopTime</td>
<td>string</td>
<td>Formatted hh:mm, using the 24-hour clock. ElectricFlow stops creating new jobs at this time, but a job in progress will continue to run. If stopTime is not specified, ElectricFlow creates one job only on each specified day.</td>
</tr>
<tr>
<td>subject</td>
<td>string</td>
<td>Refers to the object the event concerns (similar to container).</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>subjectName</td>
<td>string</td>
<td>The name of the subject/object.</td>
</tr>
<tr>
<td>subjob</td>
<td>string</td>
<td>The name of the subjob.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>string</td>
<td>The name of the nested procedure called when a step runs. If a subprocedure is specified, command or commandFile options are not necessary.</td>
</tr>
<tr>
<td>subproject</td>
<td>string</td>
<td>If a subprocedure argument was used, this is the name of the project where that subprocedure is found. By default, the current project is used.</td>
</tr>
<tr>
<td>substartingState</td>
<td>string</td>
<td>Name of the starting state for the workflow launched when the state is entered.</td>
</tr>
<tr>
<td>subworkflow</td>
<td>string</td>
<td>The name of the subworkflow.</td>
</tr>
<tr>
<td>subworkflowDefinition</td>
<td>string</td>
<td>The name of the subworkflow definition.</td>
</tr>
<tr>
<td>targetState</td>
<td>string</td>
<td>The target state for the transition definition.</td>
</tr>
<tr>
<td>testResult</td>
<td>enum</td>
<td>Possible values are: success</td>
</tr>
<tr>
<td>time</td>
<td>date</td>
<td>The time of day to invoke this schedule's procedure (24-hour clock, for example, 17:00). For a logEntry response, time indicates the time at which data was written to the log.</td>
</tr>
<tr>
<td>timeLimit</td>
<td>number</td>
<td>The maximum length of time the step is allowed to run. After the time specified, the step will be aborted. The time limit is specified in units that can be hours, minutes, or seconds.</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>enum</td>
<td>Possible values are: hours</td>
</tr>
<tr>
<td>timeout</td>
<td>number</td>
<td>Specifies the timeout for the element flag. The default value is 120 seconds.</td>
</tr>
<tr>
<td>timeZone</td>
<td>string</td>
<td>The time zone specified to use for this schedule (Java-compatible string).</td>
</tr>
<tr>
<td>totalCallCount</td>
<td>number</td>
<td>The total number of API calls to the server since startup.</td>
</tr>
<tr>
<td>totalWaitTime</td>
<td></td>
<td>On a job, this is the sum of total time all job steps waited for license, resource, and/or workspace availability.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>transitionDefinitionId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this transition definition.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>string</td>
<td>The name of the transition definition.</td>
</tr>
<tr>
<td>transitionId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this transition object.</td>
</tr>
<tr>
<td>transitionName</td>
<td>string</td>
<td>The name of the transition.</td>
</tr>
<tr>
<td>trigger</td>
<td>enum</td>
<td>Possible values are: onEnter</td>
</tr>
</tbody>
</table>
| trusted                 | boolean   | $<Boolean\ flag\ -\ 0|1|true|false>$ If "true", the resource is trusted. A trusted agent is one that has been "certificate verified." Agents can be either trusted or untrusted:
  - trusted - the ElectricFlow server verifies the agent's identity using SSL certificate verification.
  - untrusted - the ElectricFlow server does not verify agent identity. Potentially, an untrusted agent is a security risk. |
| type                    | string    | The "type" is any string value. Used primarily by the web interface to represent custom form elements. However, if "credential" is the string value, the server will expect a credential as the parameter value. |
| url                     | string    | For directory providers:
The server URL is in the form protocol://host:port//basedn.
Protocol is either ldap or ldaps (for secure LDAP).
The port is implied by the protocol, but can be overridden if it is not at the default location (389 for ldap, 636 for ldaps). The basedn is the path to the top-level directory that contains users and groups at this site. This is typically the domain name where each part is listed with a dc= and separated by commas.
Note: Spaces in the basedn must be URL encoded (%20).
For artifact repositories:
The server URL is in the form protocol://host:port/.
Typically, the repository server is configured to listen on port 8200 for https requests, so a typical URL looks like https://host:8200/. |
<table>
<thead>
<tr>
<th>Returned element</th>
<th>Type</th>
<th>Description/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>userAuthenticationTest</td>
<td>subcontainer</td>
<td>For the testDirectoryProvider API, this element authenticates the user.</td>
</tr>
<tr>
<td>userBase</td>
<td>string</td>
<td>The string prepended to the basedn to construct the directory DN that contains user records.</td>
</tr>
<tr>
<td>userId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for the user.</td>
</tr>
<tr>
<td>userInfo</td>
<td></td>
<td>findUserTest container element includes a userList subcontainer that may include multiple userInfo tags, each of which describes a user (including full name, email address, and provider name).</td>
</tr>
<tr>
<td>userList</td>
<td>list</td>
<td>findUserTest container element includes a userList subcontainer that may include one or more userInfo tags.</td>
</tr>
<tr>
<td>userName</td>
<td>string</td>
<td>The full name of the user. For Active Directory and LDAP, the name may be user@domain.</td>
</tr>
<tr>
<td>userNameAttribute</td>
<td>string</td>
<td>The attribute in a user record that contains the user's account name.</td>
</tr>
<tr>
<td>userSearchFilter</td>
<td>string</td>
<td>The LDAP query performed in the context of the user directory to search for a user by account name. The string &quot;{0}&quot; is replaced with the user's login ID. Typically, the query compares a user record attribute with the substituted user login ID.</td>
</tr>
<tr>
<td>userSearchSubtree</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>userSettingsId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for the user settings.</td>
</tr>
<tr>
<td>useSSL</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>For a string property, this is the value of the property. For a sheet property, this argument is invalid.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>version</td>
<td>string</td>
<td>For plugin versions, the value is represented in the form: major.minor. For artifact versions, the value is represented in the form: major.minor.patch-qualifier-buildNumber</td>
</tr>
<tr>
<td>waitReason</td>
<td>string</td>
<td>Possible values are: license, resource, or workspace. Generally, this objects are unavailable, causing a longer wait time for availability.</td>
</tr>
<tr>
<td>waitTime</td>
<td>number</td>
<td>The number of milliseconds the step spent between runnable and running (for example, waiting for a resource).</td>
</tr>
<tr>
<td>weekDays</td>
<td>string</td>
<td>Restricts the schedule to specified days of the week. Days of the week are separated by spaces. English names &quot;Monday&quot;, &quot;Tuesday&quot;, and so on.</td>
</tr>
<tr>
<td>workflowDefinitionId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this workflow definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>string</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for this workflow object.</td>
</tr>
<tr>
<td>workflowName</td>
<td>string</td>
<td>The name of this workflow.</td>
</tr>
<tr>
<td>workflowNameTemplate</td>
<td>string</td>
<td>Template used to determine the default names for workflows launched from a workflow definition.</td>
</tr>
<tr>
<td>workingDirectory</td>
<td>string</td>
<td>The ElectricFlow agent sets this directory as the &quot;current working directory,&quot; when running the command contained in the step. If no working directory is specified in the step, ElectricFlow uses the directory it created for the job in the ElectricFlow workspace as the working directory. <strong>Note:</strong> If running a step on a proxy resource, this directory must exist on the proxy target.</td>
</tr>
<tr>
<td>workspaceDisabled</td>
<td>boolean</td>
<td>&lt;Boolean flag - 0</td>
</tr>
<tr>
<td>workspaceId</td>
<td>number</td>
<td>The unique ElectricFlow-generated ID for the workspace.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>string</td>
<td>The name of the workspace.</td>
</tr>
<tr>
<td>Returned element</td>
<td>Type</td>
<td>Description/Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>workspaceWaitTime</td>
<td></td>
<td>The total time a job step waited for workspace availability. On a job, this is the sum of time all job steps waited for workspace availability.</td>
</tr>
<tr>
<td>zoneId</td>
<td>number</td>
<td>The ElectricFlow-generated ID for this zone.</td>
</tr>
<tr>
<td>zoneName</td>
<td>string</td>
<td>The name of the zone.</td>
</tr>
</tbody>
</table>
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>:8443/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>:8443/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. To resolve this issue, do these steps:

**Note:** This procedure has examples of messages that appear in the Google Chrome browser.

This error message appears in Google Chrome when you first enter


1. Click **Advanced**.
   
   Another error message appears.
Using the ElectricFlow RESTful API

Your connection is not private

Attackers might be trying to steal your information from nnn.nnn.nnn (for example, passwords, messages, or credit cards).

Hide advanced

You attempted to reach nnn.nnn.nnn, but the server presented a certificate issued by an entity that is not trusted by your computer’s operating system. This may mean that the server has generated its own security credentials, which Chrome cannot rely on for identity information, or an attacker may be trying to intercept your communications.

Proceed to 192.168.5.138 (unsafe)
2. Click Proceed to `<ElectricFlow_server_IP_address>` (unsafe).

The Electric Cloud API UI page opens with an error message at the top of the page:

![Error Message](image1)

3. To fix this, click in any row (for example, the artifact row), and then click Raw.

The browser window re-opens with an error message:

Unable to read api 'zone' from path `https://<ElectricFlow_server_hostname>:8443/rest/doc/v1.0/api/zone` (server returned undefined)
4. Click **Advanced**.

Another error message appears.

![Error message](image)

Your connection is not private

Attackers might be trying to steal your information from `nnn.nnn.nnn` (for example, passwords, messages, or credit cards).

You attempted to reach `nnn.nnn.nnn`, but the server presented a certificate issued by an entity that is not trusted by your computer's operating system. This may mean that the server has generated its own security credentials, which Chrome cannot rely on for identity information, or an attacker may be trying to intercept your communications.

[Proceed to 192.168.5.138 (unsafe)]

The problem is that the certificate is not trusted by the browser even though the message indicates that the certificate is not trusted by the operating system.

5. Click **Proceed to `<ElectricFlow_server_server_hostname>` (unsafe)**. The file called `<resource_name>` is downloaded.

6. Double-click the downloaded file to open it.

   The Open File dialog box opens.

7. Click **Cancel** to close the dialog box.
8. Go back to the browser page in Step 4, and click the back arrow.

The ElectricCloud API UI page still have the error message.

```
<table>
<thead>
<tr>
<th>acl</th>
<th>List Operations</th>
<th>Expand Operations</th>
<th>Raw</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclEntry</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>actualParameter</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>application</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>applicationTier</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>artifact</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>artifactVersion</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>component</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>credential</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>directoryProvider</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>emailConfig</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>emailNotifier</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>entityChange</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>environment</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>environmentInventoryItem</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>environmentTier</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>eventSubscription</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
<tr>
<td>formalParameter</td>
<td>List Operations</td>
<td>Expand Operations</td>
<td>Raw</td>
</tr>
</tbody>
</table>
```
9. Refresh (reload) the page.

The error message on the top of the page disappears.

Using the RESTful API

In the Electric Cloud API UI page:

1. Select a resource and click on the resource name.

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.
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.
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:
After clicking **Try it out!**, the response is a Request URL:

https://localhost-100.electric-cloud.com:8443/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:

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:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
<th>Parameter Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>default</td>
<td>Name for the project; must be unique among all projects.</td>
<td>path</td>
<td>string</td>
</tr>
<tr>
<td>scheduleName</td>
<td>mySchedule</td>
<td>Name for the schedule; must be unique among all schedules for the project.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>actualParameters</td>
<td></td>
<td>Parameters passed to the invoked procedure/process/workflow.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>applicationName</td>
<td></td>
<td>The name of the application that owns the process.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>beginDate</td>
<td></td>
<td>The date when this schedule will begin (for example, 2006-05-15).</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>credentialName</td>
<td></td>
<td>The name of the credential to use for impersonation.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>description</td>
<td></td>
<td>Comment text describing this object; not interpreted at all by ElectricCommander.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>endDate</td>
<td></td>
<td>The date when this schedule will end (for example, 2006-05-15). The end date is not included in the range of dates.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>interval</td>
<td></td>
<td>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.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>intervalUnits</td>
<td></td>
<td>Units for the interval of rescheduling.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>schedulePolicy</td>
<td></td>
<td>Specifies the schedule policy for a schedule.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>monthDays</td>
<td></td>
<td>A list of numbers from 1-31 separated by spaces, indicating zero or more days of the month.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>priority</td>
<td></td>
<td>The priority of the job.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>procedureName</td>
<td>Procedure</td>
<td>The name of the procedure to invoke.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>processName</td>
<td></td>
<td>The name of the application process to invoke.</td>
<td>query</td>
<td>string</td>
</tr>
<tr>
<td>scheduleDisabled</td>
<td></td>
<td>True means this schedule will not run, regardless of its settings.</td>
<td>query</td>
<td>boolean</td>
</tr>
<tr>
<td>startTime</td>
<td></td>
<td>The time of day to begin invoking this schedule’s procedure/process/workflow (24-hour clock, for example, 17:00).</td>
<td>query</td>
<td>string</td>
</tr>
</tbody>
</table>

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:8443/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 760
Common Use Cases on page 761
Generating a DSL Script for an Existing ElectricFlow Object on page 761
Passing Parameters or Arguments to Your DSL Script on page 763
Evaluating Your DSL Script in a Specific Application Context on page 763
Debugging Your DSL Script on page 764

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 670 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 670 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:

   ectool generateDsl /resources/local

   This command generates output that looks similar to the following:

   <response>
   <value>
       resource() {
           resourceName = '''local'''
           hostName = '''192.168.10.10'''
           hostType = '''CONCURRENT'''
           resourceDisabled = '''0'''
           trusted = '''0'''
           useSSL = '''1'''
           zoneName = '''default'''
       }
   </value>
   </response>

2. Copy the contents within the <response> and </response> tags to a text file and save it (for example, myScript.dsl). The copied content is the DSL script for the local resource.

   The script file contents should be similar to the following:

   resource() {
       resourceName = '''local'''
       hostName = '''192.168.10.10'''
       hostType = '''CONCURRENT'''
       resourceDisabled = '''0'''
       trusted = '''0'''
       useSSL = '''1'''
       zoneName = '''default'''
   }
3. 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 to 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:

```java
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:

```bash
ectool evalDsl --dslFile myScript.dsl --parameters '{"resourceName":"MyFirstResource", "resourceIP":"192.168.10.12"}'
```

### 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.

```bash
// 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')
    }
}
```
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.

```bash
ectool evalDsl --dslFile myScript.dsl --debug 1
```

or

```bash
ectool evalDsl <dsl text> --debug 1
```

**DSL Methods**

ElectricFlow supports these DSL methods:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>aclEntry</td>
<td>An individual access control list entry (ACE) that allows or denies a privilege on a domain object.</td>
</tr>
<tr>
<td>actualParameter</td>
<td>A name/value pair that is passed to a procedure when it is invoked.</td>
</tr>
<tr>
<td>application</td>
<td>An ElectricFlow object that you model and deploy to build, test, deploy, and release your software for continuous delivery.</td>
</tr>
<tr>
<td>applicationTier</td>
<td>A logical grouping of components that are part of an application and the resources on which they should be deployed.</td>
</tr>
<tr>
<td>artifact</td>
<td>An artifact is a top-level object containing artifact versions, a name template for published artifact versions, artifact specific properties, and access control entries to specify privileges.</td>
</tr>
<tr>
<td>artifactVersion</td>
<td>An artifact version is a collection of 0 to N files that were published to an artifact repository.</td>
</tr>
<tr>
<td>component</td>
<td>An object that is based on a specific version of an artifact and is defined in an application.</td>
</tr>
<tr>
<td>credential</td>
<td>A credential is an object that stores a user name and password for later use. You can use credentials for user impersonation and saving passwords for use inside steps.</td>
</tr>
<tr>
<td></td>
<td>It is usually used in an agent context to authenticate with a third-party system.</td>
</tr>
<tr>
<td></td>
<td>The password value is not available in the web context for security reasons, so this object has no <code>getPassword</code> method.</td>
</tr>
<tr>
<td>directoryProvider</td>
<td>Contains information about the configuration used to communicate with an external directory service (LDAP or ActiveDirectory).</td>
</tr>
<tr>
<td>emailConfig</td>
<td>Encapsulates all of the mail server configuration information necessary for the ElectricFlow server to send an e-mail message.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------</td>
</tr>
<tr>
<td>emailNotifier</td>
<td>E-mail notification to be sent when a particular event occurs.</td>
</tr>
<tr>
<td>environment</td>
<td>The environment to which an application is deployed.</td>
</tr>
<tr>
<td>environmentTemplate</td>
<td>A template defining an environment that can be spun up when the application is deployed.</td>
</tr>
<tr>
<td>environmentTemplateTier</td>
<td>A logical grouping of resources in an environment template.</td>
</tr>
<tr>
<td>environmentTemplateTierMap</td>
<td>A map that contains the mapping of application tiers to the corresponding environment template tiers.</td>
</tr>
<tr>
<td>environmentTier</td>
<td>A logical grouping of resources in an environment.</td>
</tr>
<tr>
<td>formalParameter</td>
<td>An unbound parameter defined on a procedure, workflow definition, and so on.</td>
</tr>
<tr>
<td>gateway</td>
<td>A secure connection between two zones for sharing or transferring information between the zones.</td>
</tr>
<tr>
<td>group</td>
<td>A group of users.</td>
</tr>
<tr>
<td>hook</td>
<td>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.</td>
</tr>
<tr>
<td>job</td>
<td>An instance of a procedure run.</td>
</tr>
<tr>
<td>jobStep</td>
<td>A step in a job.</td>
</tr>
<tr>
<td>license</td>
<td>License data in XML format that describes the usage to which you are entitled.</td>
</tr>
</tbody>
</table>
| pipeline              | An ElectricFlow object that orchestrates a deployment or automation.  
  ElectricFlow supports these types of pipelines:  
  - Release pipeline—Only for an application release.  
  - Generic pipeline—For any deployment or automation. |
<p>| plugin                | An add-on program used by ElectricFlow to integrate with third-party tools, custom dashboards, and unique user experiences based on roles. |
| procedure             | A container for steps that execute a task.                  |
| process               | An application or component process.                        |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processDependency</td>
<td>A dependency between process steps.</td>
</tr>
<tr>
<td>processStep</td>
<td>A step in an application or component process.</td>
</tr>
<tr>
<td>project</td>
<td>A project is a top-level container for related procedures, workflows, schedules, jobs, and properties, which is used to isolate different user groups or functions, and also encapsulate shared facilities.</td>
</tr>
<tr>
<td>property</td>
<td>A name-value pair associated with ElectricFlow objects to provide additional information beyond what is already built into the system. Built-in data is also accessible through the property mechanism. ElectricFlow supports intrinsic and custom properties.</td>
</tr>
<tr>
<td>repository</td>
<td>An object that stores artifact versions. It primarily contains information about how to connect to a particular artifact repository.</td>
</tr>
<tr>
<td>resource</td>
<td>An agent machine that is configured to communicate with ElectricFlow and where job steps can be executed.</td>
</tr>
<tr>
<td>resourcePool</td>
<td>A collection of resources with an ordering policy.</td>
</tr>
<tr>
<td>resourceTemplate</td>
<td>A template with the required information to provision and later spin up cloud resources on an on-demand basis.</td>
</tr>
<tr>
<td>schedule</td>
<td>An object that launches a procedure at a specified time in the future, possibly on a regular interval.</td>
</tr>
<tr>
<td>snapshot</td>
<td>A version of an application with specific artifact versions and at a specific state at any point in time.</td>
</tr>
<tr>
<td>stage</td>
<td>A logical grouping of pipeline tasks.</td>
</tr>
<tr>
<td>stateDefinition</td>
<td>A state definition in a workflow definition. Each workflow can contain one or more states.</td>
</tr>
<tr>
<td>step</td>
<td>A unit of logic that will execute on an agent.</td>
</tr>
<tr>
<td>task</td>
<td>A representation of task in a stage or gate.</td>
</tr>
<tr>
<td>tierMap</td>
<td>A map that contains the mapping of application tiers to the corresponding environment tiers.</td>
</tr>
<tr>
<td>transitionDefinition</td>
<td>How a workflow transitions from one state to another.</td>
</tr>
<tr>
<td>user</td>
<td>A user defines an account used to log into the system and control access to ElectricFlow objects.</td>
</tr>
</tbody>
</table>
### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workflowDefinition</td>
<td>Workflow objects are split into two types: Definition objects and Instance objects. Definition objects provide the template for a running workflow instance.</td>
</tr>
<tr>
<td>workspace</td>
<td>A workspace is a subtree of files and directories where job file data is stored. The term &quot;workspace&quot; typically refers to the top-level directory in this subtree.</td>
</tr>
<tr>
<td>zone</td>
<td>A zone or top-level network created as a way to partition a collection of agents to secure them from use by other groups.</td>
</tr>
</tbody>
</table>

### aclEntry

An individual access control list entry (ACE) that allows or denies a privilege on a domain object.

#### Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>principalName</td>
<td>Name of the user or group for this access control entry.</td>
</tr>
<tr>
<td>principalType</td>
<td>Type of principal for this access control entry (user or group).</td>
</tr>
</tbody>
</table>

#### Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifactVersion container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>changePermissionsPrivilege</td>
<td>Determines whether the principal can modify access control for the object.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>The name of the environment template container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>The name of the environment template tier container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>The name of the environment template container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>The name of the environment tier container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>executePrivilege</td>
<td>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.</td>
</tr>
<tr>
<td>flowName</td>
<td>The name of the flow container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>flowStateName</td>
<td>The name of the flow state container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>flowTransitionName</td>
<td>The name of the flow transition container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway container of the property sheet.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>jobId</td>
<td>The primary key or name of the job container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The primary key of the job-step container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>modifyPrivilege</td>
<td>Determines whether the principal can change the contents of the object.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the notifier container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>objectId</td>
<td>The object id as returned by <code>findObjects</code>.</td>
</tr>
<tr>
<td>path</td>
<td>Property path string.</td>
</tr>
<tr>
<td>pipelineName</td>
<td>The name of the pipeline container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The primary key of the property sheet that owns the property.</td>
</tr>
<tr>
<td>readPrivilege</td>
<td>Determines whether the principal can examine the contents of the object</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of the resource pool container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>The name of the resource template container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule container of the property sheet.</td>
</tr>
<tr>
<td>snapshotName</td>
<td>The name of the snapshot container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>stageName</td>
<td>The name of the stage container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition container of the property sheet that owns the property.</td>
</tr>
</tbody>
</table>
## Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateName</td>
<td>The name of the state container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>systemObjectName</td>
<td>The system object.</td>
</tr>
<tr>
<td>taskName</td>
<td>The name of the task that owns property sheet.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the transition definition container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>transitionName</td>
<td>The name of the transition container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>userName</td>
<td>The name of the user container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow container of the property sheet that owns the property.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace container of the property sheet.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The name of the zone container of the property sheet.</td>
</tr>
</tbody>
</table>

### actualParameter

A name/value pair that is passed to a procedure when it is invoked.

#### Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameterName</td>
<td>The name of the parameter to create, modify, or delete.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

#### Optional Arguments
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application, if the actual parameter is on an application process step.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component, if the actual parameter is on a component process step.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the actual parameter is on a process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the actual parameter is on a process step.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>value</td>
<td>The value of the actual parameter, if creating or modifying.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**application**

An ElectricFlow object that you model and deploy to build, test, deploy, and release your software for continuous delivery.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

**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 that is being renamed.  

**DSL Methods for ElectricFlow Objects That Can be Nested Inside**
- `component`
- `applicationTier`
- `emailNotifier`
- `environmentTemplateTierMap`
- `property`
- `tierMap`
- `snapshot`
- `process`

**applicationTier**
A logical grouping of a components that are part of an application and the resources on which they should be deployed.
When `applicationTier` has a nested `component`, DSL automatically calls the `addComponentToApplicationTier` API.

**Required Arguments**

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

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>description</code></td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td><code>newName</code></td>
<td>New name for an existing object.</td>
</tr>
</tbody>
</table>
DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

artifact

A top-level object containing artifact versions, a name template for published artifact versions, artifact specific properties, and access control entries to specify privileges.

Required Arguments
None.

Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactKey</td>
<td>The artifactKey component of the GroupId/ArtifactKey/Version (GAV) coordinates.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact.</td>
</tr>
<tr>
<td>artifactVersionNameTemplate</td>
<td>The artifactVersion name template.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>groupId</td>
<td>The groupId component of the GroupId/ArtifactKey/Version (GAV) coordinates.</td>
</tr>
</tbody>
</table>

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- artifactVersion
- property

artifactVersion

A collection of 0 to N files that were published to an artifact repository.

Required Arguments
None.

Optional Arguments
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactKey</td>
<td>The artifactKey component of the GroupId/ArtifactKey/Version (GAV) coordinates.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact containing the artifactVersion.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifactVersion.</td>
</tr>
<tr>
<td>artifactVersionState</td>
<td>The state of the artifactVersion.</td>
</tr>
<tr>
<td>dependentArtifactVersions</td>
<td>The set of artifactVersions on which this artifactVersion depends.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>groupId</td>
<td>The groupId component of the GAV (GroupId/ArtifactVersionId/Version) coordinates.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The id of the job step; used to make a project association.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>removeAllDependentArtifactVersions</td>
<td>If this is set to true or 1, all dependencies will be removed.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the artifact repository.</td>
</tr>
<tr>
<td>version</td>
<td>The version component of the GAV (GroupId/ArtifactVersionId/Version) coordinates.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- `property`

**component**

An object that is based on a specific version of an artifact and is defined in an application.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentName</td>
<td>The name of the component</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>
Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of an application in which the component is defined.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of a credential to attach to this component.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>pluginKey</td>
<td>The key of the plugin.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin.</td>
</tr>
<tr>
<td>pluginParameters</td>
<td>List of plugin parameters.</td>
</tr>
<tr>
<td></td>
<td>An alternate name is pluginParameter.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- `property`
- `process`

**credential**

A credential is an object that stores a user name and password for later use. You can use credentials for user impersonation and saving passwords for use inside steps.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>The name of the credential.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
newName | New name for an existing object that is being renamed.
password | The password for the credential. It can also be a certificate or a chunk of data.
passwordRecoveryAllowed | If set to `true` or `1`, ElectricFlow recovers the password by invoking `getFullCredential` from a job step.
userName | The user name for the credential.

dsl Methods for ElectricFlow Objects That Can Be Nested Inside

- `property`

### directoryProvider

Contains information about the configuration used to communicate with an external directory service (LDAP or ActiveDirectory).

#### Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>providerName</td>
<td>Name for a LDAP directory provider that must be unique.</td>
</tr>
</tbody>
</table>

#### Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>commonGroupNameAttribute</td>
<td>The attribute in a group record that contains the common name of the group.</td>
</tr>
<tr>
<td></td>
<td>If specified, it is only used when searching for groups from an external provider.</td>
</tr>
<tr>
<td></td>
<td>It is usually used when the group name attribute is set to <code>distinguishedName</code>, because this field is not searchable.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>domainName</td>
<td>The domain from which the Active Directory servers are automatically discovered.</td>
</tr>
<tr>
<td>emailAttribute</td>
<td>The attribute in a LDAP user record that contains the user's email.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>enableGroups</td>
<td>Determines whether or not to enable external groups for the directory provider.</td>
</tr>
<tr>
<td>fullUserNameAttribute</td>
<td>The attribute in a user record that contains the user's full name (first and last).</td>
</tr>
<tr>
<td>groupBase</td>
<td>String prepended to the base distinguished name (DN) to construct the DN of the directory that contains group records.</td>
</tr>
<tr>
<td>groupMemberAttributes</td>
<td>Comma separated list of attribute names that can identify a member of a group.</td>
</tr>
<tr>
<td>groupMemberFilter</td>
<td>LDAP query string for the groups directory to find groups that contain a given user as a member.</td>
</tr>
<tr>
<td>groupNameAttribute</td>
<td>The attribute in a group record that contains the name of the group.</td>
</tr>
<tr>
<td>groupSearchFilter</td>
<td>LDAP query string used in group directory to enumerate group records.</td>
</tr>
<tr>
<td>managerDn</td>
<td>The name of a user who has read-only access to the LDAP or Active Directory server. It is usually a distinguished name (DN). A simple name may be used when the Active Directory server URL is being auto-discovered via DNS.</td>
</tr>
<tr>
<td>managerPassword</td>
<td>Secret value used to identify the account for the query user.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>providerType</td>
<td>Type string for a directory provider: ldap or activedirectory.</td>
</tr>
<tr>
<td>realm</td>
<td>The realm of the LDAP directory provider. This is used to create unique user names when there are multiple providers.</td>
</tr>
<tr>
<td>url</td>
<td>The URL of the LDAP Directory Provider server.</td>
</tr>
<tr>
<td>useSSL</td>
<td>If this argument is set to true or 1, SSL is used for communication.</td>
</tr>
<tr>
<td>userBase</td>
<td>Used to construct the distinguished name (DN) of the directory that contain user records.</td>
</tr>
<tr>
<td>userNameAttribute</td>
<td>The attribute in a user record that contains the user's account name.</td>
</tr>
<tr>
<td>userSearchFilter</td>
<td>RFC 2254 LDAP query to search for a user by name.</td>
</tr>
<tr>
<td>userSearchSubtree</td>
<td>If this argument is set to true or 1, ElectricFlow recursively searches the subtree below the user base.</td>
</tr>
</tbody>
</table>
DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

emailConfig

Encapsulates all of the mail server configuration information necessary for the ElectricFlow server to send an email message.

Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>configName</td>
<td>The email configuration name.</td>
</tr>
</tbody>
</table>

Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>mailFrom</td>
<td>The email address used as the email sender address for ElectricFlow notifications.</td>
</tr>
<tr>
<td>mailHost</td>
<td>Name of the email server host.</td>
</tr>
<tr>
<td>mailPort</td>
<td>The port number for the email service on the server.</td>
</tr>
<tr>
<td>mailProtocol</td>
<td>Name of the email transport protocol. Supported protocol names are SMTP and SMTPS.</td>
</tr>
<tr>
<td>mailUser</td>
<td>Name of the email user on behalf of which ElectricFlow sends email notifications.</td>
</tr>
<tr>
<td>mailUserPassword</td>
<td>Password of the email user on behalf of which ElectricFlow sends email notifications.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
</tbody>
</table>

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

emailNotifier

Email notification to be sent when a particular event occurs.
Using the ElectricFlow DSL

### Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifierName</td>
<td>The name of the email notifier.</td>
</tr>
</tbody>
</table>

### Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application that is related to the target email container, a process or process step.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component which is related to the target email container, a process or process step.</td>
</tr>
<tr>
<td>condition</td>
<td>A fixed text or text embedding property references that is evaluated into a logical TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE.</td>
</tr>
<tr>
<td>configName</td>
<td>Name for an email configuration, or text that through property expansion results in such an email configuration name.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>destinations</td>
<td>A list of space-separated user names, email addresses, or email aliases or text that through property expansion results in such a list.</td>
</tr>
<tr>
<td>environmentNames</td>
<td>Name of the environments. An alternate argument name is environmentName.</td>
</tr>
<tr>
<td>eventType</td>
<td>An enumeration of valid event categories recognized by the email notifiers.</td>
</tr>
<tr>
<td>formattingTemplate</td>
<td>String containing the email formatting instructions for generating notifications.</td>
</tr>
<tr>
<td>gateType</td>
<td>The type of the gate.</td>
</tr>
<tr>
<td>groupNames</td>
<td>A list of names of the groups which receives the notification. An alternate argument name is groupName.</td>
</tr>
<tr>
<td>jobId</td>
<td>The primary key or name of the job container of the email notifier.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The primary key of the job-step container of the email notifier.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>notificationType</td>
<td>The notification type that will be stored in the ec_notificationType property.</td>
</tr>
<tr>
<td>pipelineName</td>
<td>The name of the pipeline container of the email notifier.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure container of the email notifier.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process container of the email notifier.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step container of the email notifier.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project container of the email notifier.</td>
</tr>
<tr>
<td>stageName</td>
<td>The name of the stage container of the email notifier.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition container of the email notifier.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state container of the email notifier.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step container of the email notifier.</td>
</tr>
<tr>
<td>taskName</td>
<td>The name of the task container of the email notifier.</td>
</tr>
<tr>
<td>userNames</td>
<td>A list of names of the users who receives the notification.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>An alternate argument name is userName.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of the workflow definition container of the email notifier.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- **property**

**environment**

The environment to which an application is deployed.

**Required Arguments**
### Using the ElectricFlow DSL

#### Name | Description
--- | ---
environmentName | The name of the environment.
projectName | Name for the project that must be unique among all projects.

#### Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>Create an environment from the specified application.</td>
</tr>
<tr>
<td>applicationProjectName</td>
<td>The application project name.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>environmentEnabled</td>
<td>If this argument is set to true or 1, the environment is enabled.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an object that is being renamed.</td>
</tr>
</tbody>
</table>

#### DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- environmentTier
- property

### environmentTemplate

A template defining an environment that can be spun up when the application is deployed.

#### Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentTemplateName</td>
<td>The name of the environment template.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

#### Optional Arguments
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an object that is being renamed.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- **property**
- **environmentTemplateTier**

### environmentTemplateTier

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentTemplateName</td>
<td>The name of the environment template.</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>Name for the environment template tier that must be unique among all tiers for the environment template.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an object that is being renamed.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- **property**
environmentTemplateTierMap

A map that contains the mapping of application tiers to the corresponding environment template tiers.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application to which the environment template tiers are mapped.</td>
</tr>
<tr>
<td>environmentProjectName</td>
<td>The name of the environment project name.</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>The name of the environment template.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationEntityRevisionId</td>
<td>Revision ID of the versioned object.</td>
</tr>
<tr>
<td>tierMapName</td>
<td>The name of the environment template tier map.</td>
</tr>
<tr>
<td></td>
<td>If this is not specified, ElectricFlow uses a hyphenated name consisting of</td>
</tr>
<tr>
<td></td>
<td>the application and environment names.</td>
</tr>
<tr>
<td>tierMappings</td>
<td>The list of mappings between the application tiers and the environment</td>
</tr>
<tr>
<td></td>
<td>template tiers. An alternate argument name is 'tierMapping'.</td>
</tr>
</tbody>
</table>

environmentTier

A logical grouping of resources in an environment.

When `environmentTier` has a nested resource, DSL automatically calls the `addResourceToEnvironmentTier` API.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentName</td>
<td>The name of the environment.</td>
</tr>
</tbody>
</table>
### Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>environmentTierName</td>
<td>Name for the environment tier that must be unique among all tiers for the environment.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

### Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an object that is being renamed.</td>
</tr>
</tbody>
</table>

### DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- `property`

#### formalParameter

An unbound parameter defined on a procedure, workflow definition, and so on.

### Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formalParameterName</td>
<td>Name for this parameter; used when the procedure is invoked to specify a value for the parameter.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

### Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application, if the formal parameter is on an application process.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component, if the formal parameter is on a component process.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>defaultValue</td>
<td>If no value is provided for the parameter when the procedure is invoked, this value will be used.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>expansionDeferred</td>
<td>True means 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.</td>
</tr>
<tr>
<td>flowName</td>
<td>The name of the flow to which the flow state belongs to.</td>
</tr>
<tr>
<td>flowStateName</td>
<td>The name of the flow state, if the formal parameter is on a flow state.</td>
</tr>
<tr>
<td>label</td>
<td>Specifies the display label.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>orderIndex</td>
<td>Specifies the display order index (starts from 1).</td>
</tr>
<tr>
<td>pipelineName</td>
<td>The name of the pipeline, if the formal parameter is on a pipeline.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the formal parameter is on a process.</td>
</tr>
<tr>
<td>required</td>
<td>True means this parameter is required: the procedure will not execute unless a value is given for the parameter.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of a workflow state.</td>
</tr>
<tr>
<td>type</td>
<td>The type of a formal parameter.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
<tr>
<td>workflowName</td>
<td>The name of a workflow.</td>
</tr>
</tbody>
</table>

gateway

A secure connection between two zones for sharing or transferring information between the zones.

**Required Arguments**
### Name | Description
---|---
gatewayName | The gateway name.

groupName | Name of the group that must be unique among local groups.

group

**A group of users.**

**Required Arguments**
Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>migrateSettings</td>
<td>New group name to which settings will be migrated.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>removeAllUsers</td>
<td>True to remove all users from this group.</td>
</tr>
<tr>
<td>userNames</td>
<td>List of users in the group. (Alternate argument name 'userName')</td>
</tr>
</tbody>
</table>

DSL Methods for ElectricFlow Objects That Can Be Nested Inside
- property

**hook**

A resource template hook that stores a reference to a procedure in an ElectricFlow project or plugin project. When a resource template is used to create a resource pool, these procedures are invoked.

Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hookName</td>
<td>Name for the hook that must be unique among all hookd.</td>
</tr>
</tbody>
</table>

Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>broadcast</td>
<td>broadcast flag</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>hookParameters</td>
<td>hook parameters (Alternate argument name 'hookParameter')</td>
</tr>
<tr>
<td>hookType</td>
<td>hook type</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>procedureName</td>
<td>hook procedure name</td>
</tr>
<tr>
<td>procedurePluginKey</td>
<td>procedure plugin key</td>
</tr>
</tbody>
</table>
### DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- **property**

---

**job**

An instance of a procedure run.

**Required Arguments**

None.

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>destinationProject</td>
<td>The project that will own the job.</td>
</tr>
<tr>
<td>jobId</td>
<td>The primary key of the job, or the name of the job.</td>
</tr>
<tr>
<td>jobNameNameTemplate</td>
<td>Template used to determine the default name of jobs launched from a procedure.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that should 'own' the job step. If not specified, myStep.procedure is used.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project is destinationProject is not specified.</td>
</tr>
<tr>
<td>status</td>
<td>The starting status for the job.</td>
</tr>
</tbody>
</table>

---

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- **jobStep**
- **emailNotifier**
- **property**
**jobStep**

A step in a job.

**Required Arguments**

None.

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameters</td>
<td>Actual parameters passed to an invoked subprocedure (Alternate argument name 'actualParameter')</td>
</tr>
<tr>
<td>alwaysRun</td>
<td>True means this step will run even if preceding steps fail in a way that aborts the job</td>
</tr>
<tr>
<td>broadcast</td>
<td>True means replicate this step 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).</td>
</tr>
<tr>
<td>command</td>
<td>Script to execute the functions of this step; passed to the step's shell for execution.</td>
</tr>
<tr>
<td>condition</td>
<td>A fixed text or text embedding property references that is evaluated into a logical TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential to be used for impersonation.</td>
</tr>
<tr>
<td>credentials</td>
<td>The list of runtime credentials attached to the job step. (Alternate argument name 'credential')</td>
</tr>
<tr>
<td>errorHandling</td>
<td>Specifies error handling for this step.</td>
</tr>
<tr>
<td>exclusive</td>
<td>True means the resource acquired for this step will be retained for the exclusive use of this job. This means 2 things: first, no other job will be able to use that resource, regardless of its step limit, until this job completes; second, 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.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>exclusiveMode</td>
<td>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.</td>
</tr>
<tr>
<td>external</td>
<td>True if the step is externally managed (no state machine).</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The primary key of the job step</td>
</tr>
<tr>
<td>jobStepName</td>
<td>The name for the new step. If omitted, a default name will be generated.</td>
</tr>
<tr>
<td>logFileName</td>
<td>Name of the log file for a step; specified relative to the root directory in the job’s workspace.</td>
</tr>
<tr>
<td>parallel</td>
<td>True means this step and all adjacent steps with the flag set will run in parallel.</td>
</tr>
<tr>
<td>parentPath</td>
<td>Path to the parent job step. If a parent step is not specified, the current job step is used.</td>
</tr>
<tr>
<td>postProcessor</td>
<td>This command runs in parallel with the main command for the step; it analyzes the log for the step and collects diagnostic information.</td>
</tr>
<tr>
<td>precondition</td>
<td>A fixed text or text embedding property references that is evaluated into a logical TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure that should 'own' the job step. If not specified, myStep.procedure is used.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project for procedureName.</td>
</tr>
<tr>
<td>releaseExclusive</td>
<td>True means the resource acquired for this step will be no longer be retained for the exclusive use of this job when this step completes.</td>
</tr>
<tr>
<td>releaseMode</td>
<td>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.</td>
</tr>
<tr>
<td>resourceName</td>
<td>Name for the resource that must be unique among all resources.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>shell</td>
<td>Name of the shell program that will execute the command and postprocessor for the step.</td>
</tr>
<tr>
<td>status</td>
<td>The starting status for the step.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the procedure step that should 'own' the job step. If not specified, myStep is used.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>Name of a procedure to invoke during this step.</td>
</tr>
<tr>
<td>subproject</td>
<td>Name of the project containing the procedure to invoke during this step.</td>
</tr>
<tr>
<td>timeLimit</td>
<td>Maximum amount of time the step can execute; abort if it exceeds this time.</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>Units for step time limit: seconds, minutes, or hours.</td>
</tr>
<tr>
<td>workingDirectory</td>
<td>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.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- jobStep
- emailNotifier
- property

**license**

License data in XML format that describes the usage to which you are entitled.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>licenseData</td>
<td>Container elements for license data, which expects embedded XML as CDATA.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

None.
DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

pipeline

An ElectricFlow object that orchestrates a deployment or automation.

ElectricFlow supports these types of pipelines:

- Release pipeline—Only for an application release.
- Generic pipeline—For any deployment or automation.

Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipelineName</td>
<td>The name of the pipeline</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>enabled</td>
<td>True to enable the pipeline.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>type</td>
<td>Type of pipeline</td>
</tr>
</tbody>
</table>

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- formalParameter
- stage
- property

plugin

An add-on program used by ElectricFlow to integrate with third-party tools, custom dashboards, and unique user experiences based on roles.
Using the ElectricFlow DSL

Required Arguments
None.

Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>author</td>
<td>The name of the plugin author.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>key</td>
<td>Version independent name for the plugin.</td>
</tr>
<tr>
<td>label</td>
<td>Label to display in lists for the plugin.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project</td>
</tr>
<tr>
<td>version</td>
<td>The version of the plugin.</td>
</tr>
</tbody>
</table>

DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

procedure

Container for steps that execute some task.

Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>procedureName</td>
<td>Name for the procedure that must be unique within the project.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>The name of a credential to attach to this procedure.</td>
</tr>
</tbody>
</table>
### Name | Description
--- | ---
`description` | Comment text describing this object that is not interpreted at all by ElectricFlow.
`jobNameTemplate` | Template used to determine the default name of jobs launched from a procedure.
`newName` | New name for an existing object that is being renamed.
`resourceName` | The name of the default resource for this procedure.
`timeLimit` | Maximum amount of time the step can execute; abort if it exceeds this time.
`timeLimitUnits` | Units for step time limit: seconds, minutes, or hours.
`workspaceName` | The name of the default workspace for this procedure.

### DSL Methods for ElectricFlow Objects That Can Be Nested Inside
- `formalParameter`
- `emailNotifier`
- `property`

### process
An application or component process.

#### Required Arguments

### Name | Description
--- | ---
`processName` | The name of the process.
`projectName` | 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.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentApplicationName</td>
<td>If specified, the component is scoped to this application not the project.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component, if the process is owned by a component.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of a credential to attach to this process.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>processType</td>
<td>Defines type of action performed by the process.</td>
</tr>
<tr>
<td>timeLimit</td>
<td>Maximum amount of time the step can execute; abort if it exceeds this time.</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>Units for step time limit: seconds, minutes, or hours.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the default workspace for this process.</td>
</tr>
</tbody>
</table>

### DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- processStep
- formalParameter
- emailNotifier
- property
- processDependency

**processDependency**

Represents a dependency between process steps.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processName</td>
<td>The name of the process.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td>targetProcessStepName</td>
<td>The name of the target process step.</td>
</tr>
</tbody>
</table>
Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application, if the process is owned by an application.</td>
</tr>
<tr>
<td>branchCondition</td>
<td>Branch Condition.</td>
</tr>
<tr>
<td>branchConditionName</td>
<td>Branch Condition Name.</td>
</tr>
<tr>
<td>branchConditionType</td>
<td>Branch Condition Type.</td>
</tr>
<tr>
<td>branchType</td>
<td>Branch Type.</td>
</tr>
<tr>
<td>componentApplicationName</td>
<td>If specified, the component is scoped to this application not the project.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component, if the process is owned by a component.</td>
</tr>
</tbody>
</table>

processStep

A step in an application or component process.

Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processName</td>
<td>The name of the process.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameters</td>
<td>Actual parameters passed to an invoked subprocedure or process. (Alternate argument name 'actualParameter')</td>
</tr>
<tr>
<td>afterProcessStep</td>
<td>If specified, the process step will be placed after the named process step.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application, if the process is owned by an application.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>If references an application tier, the name of the application tier.</td>
</tr>
<tr>
<td>beforeProcessStep</td>
<td>If specified, the process step will be placed before the named process step.</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>True if the step should remove all actual parameters.</td>
</tr>
<tr>
<td>componentApplicationName</td>
<td>If specified, the component is scoped to this application not the project.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component, if the process is owned by a component.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential object.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>errorHandling</td>
<td>Specifies error handling for this step.</td>
</tr>
<tr>
<td>includeCompParameterRef</td>
<td>True if the actual parameters should be generated from component properties. Works for artifact components only.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>processStepType</td>
<td>Defines type of the process step.</td>
</tr>
<tr>
<td>subcomponent</td>
<td>If referencing a component process, the name of the component.</td>
</tr>
<tr>
<td>subcomponentApplicationName</td>
<td>If referencing a component process, the name of the component application (if not project scoped).</td>
</tr>
<tr>
<td>subcomponentProcess</td>
<td>If referencing a component process, the name of the component process.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>If referencing a procedure, the name of the procedure.</td>
</tr>
<tr>
<td>subproject</td>
<td>If referencing a procedure, the name of the procedure’s project.</td>
</tr>
<tr>
<td>timeLimit</td>
<td>Maximum amount of time the step can execute; abort if it exceeds this time.</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>Units for step time limit: seconds, minutes, or hours.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace.</td>
</tr>
</tbody>
</table>
**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- emailNotifier
- property

**project**

Container for a group of related procedures and schedules.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>credentialName</td>
<td>The name of the credential object.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>resourceName</td>
<td>Name for the resource that must be unique among all resources.</td>
</tr>
<tr>
<td>tracked</td>
<td>True to enable change tracking for this project.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- component
- application
- flow
- pipeline
- schedule
- credential
- process
- workflowDefinition
Using the ElectricFlow DSL

- environment
- stage
- property
- resourceTemplate
- procedure
- environmentTemplate

**property**

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>propertyName</td>
<td>Name for the property that must be unique within the property sheet.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>applicationTierName</td>
<td>The name of the application tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactName</td>
<td>The name of the artifact container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>artifactVersionName</td>
<td>The name of the artifactVersion container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>componentName</td>
<td>The name of the component container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>configName</td>
<td>The name of the emailConfig container that owns the property.</td>
</tr>
<tr>
<td>counter</td>
<td>Whether or not the property is used as a counter.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>The name of the environment template container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>environmentTemplateTierName</td>
<td>The name of the environment template tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>environmentTierName</td>
<td>The name of the environment tier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>expandable</td>
<td>Whether or not the property is recursively expandable.</td>
</tr>
<tr>
<td>extendedContextSearch</td>
<td>For simple property names, whether or not to search objects in the hierarchy to find the desired property.</td>
</tr>
<tr>
<td>flowName</td>
<td>The name of the flow container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>flowStateName</td>
<td>The name of the flow state container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>flowTransitionName</td>
<td>The name of the flow transition container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>gatewayName</td>
<td>The name of the gateway container of the property sheet.</td>
</tr>
<tr>
<td>groupName</td>
<td>The name of the group container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>jobId</td>
<td>The primary key or name of the job container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>jobStepId</td>
<td>The primary key of the job-step container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>notifierName</td>
<td>The name of the notifier container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>objectId</td>
<td>The object id as returned by FindObjects.</td>
</tr>
<tr>
<td>path</td>
<td>Property path string.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>pipelineName</td>
<td>The name of the pipeline container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>pluginName</td>
<td>The name of the plugin container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the process, if the container is a process or process step.</td>
</tr>
<tr>
<td>processStepName</td>
<td>The name of the process step, if the container is a process step.</td>
</tr>
<tr>
<td>projectName</td>
<td>The name of the project container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>propertySheetId</td>
<td>The primary key of the property sheet which owns the property.</td>
</tr>
<tr>
<td>propertyType</td>
<td>Type of property.</td>
</tr>
<tr>
<td>repositoryName</td>
<td>The name of the repository container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>resourceName</td>
<td>The name of the resource container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>resourcePoolName</td>
<td>The name of the resource pool container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>The name of the resource template container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>scheduleName</td>
<td>The name of the schedule container of the property sheet.</td>
</tr>
<tr>
<td>snapshotName</td>
<td>The name of the snapshot container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>stageName</td>
<td>The name of the stage container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name of the state definition container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>stateName</td>
<td>The name of the state container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>stepName</td>
<td>The name of the step container of the property sheet which owns the property.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>systemObjectName</code></td>
<td>The system object.</td>
</tr>
<tr>
<td><code>taskId</code></td>
<td>The name of the task which owns property sheet.</td>
</tr>
<tr>
<td><code>transitionDefinitionName</code></td>
<td>The name of the transition definition container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>transitionName</code></td>
<td>The name of the transition container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>userName</code></td>
<td>The name of the user container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>The value of the property.</td>
</tr>
<tr>
<td><code>workflowDefinitionName</code></td>
<td>The name of the workflow definition container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>workflowName</code></td>
<td>The name of the workflow container of the property sheet which owns the property.</td>
</tr>
<tr>
<td><code>workspaceName</code></td>
<td>The name of the workspace container of the property sheet.</td>
</tr>
<tr>
<td><code>zoneName</code></td>
<td>The name of the zone container of the property sheet.</td>
</tr>
</tbody>
</table>

**repository**

An object that stores artifact versions.

It primarily contains information about how to connect to a particular artifact repository.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>repositoryName</code></td>
<td>The repository name.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>description</code></td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
</tbody>
</table>
### DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- **property**

---

#### resource

An agent machine that is configured to communicate with ElectricFlow and where job steps can be executed.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourceName</td>
<td>Name for the resource that must be unique among all resources.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>artifactCacheDirectory</td>
<td>Artifact cache directory for this resource.</td>
</tr>
<tr>
<td>block</td>
<td>True to block on the agent ping before returning.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>hostName</td>
<td>The domain name or IP address of the server machine corresponding to this resource.</td>
</tr>
<tr>
<td>hostType</td>
<td>The type of the host.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>pools</td>
<td>A list of arbitrary names separated by spaces, indicating the pools with which this resource is associated.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>port</td>
<td>Port number to use when connecting to the agent for this resource; defaults to server default.</td>
</tr>
<tr>
<td>proxyCustomization</td>
<td>Proxy specific customization data; defaults to none.</td>
</tr>
<tr>
<td>proxyHostName</td>
<td>The domain name or IP address of the proxy agent machine corresponding to this resource.</td>
</tr>
<tr>
<td>proxyPort</td>
<td>Port number to use when connecting to the proxy agent for this resource; defaults to server default.</td>
</tr>
<tr>
<td>proxyProtocol</td>
<td>The protocol to use when proxying to this resource; defaults to none.</td>
</tr>
<tr>
<td>repositoryNames</td>
<td>A newline delimited list of repositories to retrieve artifacts from.</td>
</tr>
<tr>
<td>resourceDisabled</td>
<td>True means this resource will not be allocated to job steps, regardless of its step limit.</td>
</tr>
<tr>
<td>shell</td>
<td>Name of the shell program that will execute the command and postprocessor for the step.</td>
</tr>
<tr>
<td>stepLimit</td>
<td>The maximum number of steps that may execute simultaneously using this resource.</td>
</tr>
<tr>
<td>trusted</td>
<td>True means the agent can speak to all other trusted agents in its zone. An untrusted agent can only speak to gateway agents.</td>
</tr>
<tr>
<td>useSSL</td>
<td>True means SSL is used for communication.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace.</td>
</tr>
<tr>
<td>zoneName</td>
<td>Name for the zone that must be unique among all zones.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- `property`

**resourcePool**

A collection of resources with a ordering policy.

**Required Arguments**
Using the ElectricFlow DSL

### Resource Pool Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>resourcePoolName</td>
<td>Name for the resource pool that must be unique among all resource pools.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoDelete</td>
<td>If true, the pool is deleted when the last resource is deleted.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>orderingFilter</td>
<td>JavaScript fragment that returns custom ordering of resources in a pool.</td>
</tr>
<tr>
<td>resourceName</td>
<td>List of resources to add/remove from the pool. (Alternate argument name 'resourceName')</td>
</tr>
<tr>
<td>resourcePoolDisabled</td>
<td>True means the resourcePool will not be allocated to job steps.</td>
</tr>
</tbody>
</table>

### DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- `property`

---

**resourceTemplate**

A template with the required information to provision and later spin up cloud resources on an on-demand basis.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td>resourceTemplateName</td>
<td>Name for the resource template that must be unique among all resource templates.</td>
</tr>
</tbody>
</table>

**Optional Arguments**
### Name | Description
--- | ---
`cfgMgrParameters` | Configuration manager plugin parameters. (The alternate argument name 'cfgMgrParameter'.)
`cfgMgrPluginKey` | Configuration manager plugin key.
`cfgMgrProcedure` | Configuration manager plugin method name.
`cfgMgrProjectName` | Configuration manager plugin project name.
`cloudProviderParameters` | Cloud provider plugin parameters. (The alternate argument name 'cloudProviderParameter'.)
`cloudProviderPluginKey` | Cloud provider plugin key.
`cloudProviderProcedure` | Cloud provider plugin method name.
`cloudProviderProjectName` | Cloud provider plugin project name.
`description` | Comment text describing this object that is not interpreted at all by ElectricFlow.
`newName` | New name for an existing object that is being renamed.

### DSL Methods for ElectricFlow Objects That Can Be Nested Inside
- `hook`
- `property`

### schedule

This object is responsible for launching a procedure at some time in the future, possibly on a regular interval.

#### Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>projectName</code></td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td><code>scheduleName</code></td>
<td>Name for the schedule that must be unique among all schedules for the project.</td>
</tr>
</tbody>
</table>

#### Optional Arguments
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameters</td>
<td>Parameters passed to the invoked procedure/process/workflow. (Alternate argument name 'actualParameter')</td>
</tr>
<tr>
<td>applicationName</td>
<td>The name of the application that owns the process.</td>
</tr>
<tr>
<td>beginDate</td>
<td>The date when this schedule will begin (for example, 2006-05-15).</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>Whether or not to clear actual parameters for this object.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the credential to use for impersonation.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>endDate</td>
<td>The date when this schedule will end (for example, 2006-05-15). The end date is not included in the range of dates.</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment used to determine where to run the process.</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>The name of the environment template used to determine the environment where to run the process.</td>
</tr>
<tr>
<td>environmentTemplateTierMapName</td>
<td>The name of the environment template tier map used to determine how to spin up the environment which will be used to run the process.</td>
</tr>
<tr>
<td>interval</td>
<td>If specified, the procedure/process/workflow will be rescheduled over and over again at intervals of this length. &quot;Continuous!&quot; means reschedule the procedure/process/workflow as soon as the previous job finishes.</td>
</tr>
<tr>
<td>intervalUnits</td>
<td>Units for the interval of rescheduling.</td>
</tr>
<tr>
<td>misfirePolicy</td>
<td>Specifies the misfire policy for a schedule.</td>
</tr>
<tr>
<td>monthDays</td>
<td>A list of numbers from 1-31 separated by spaces, indicating zero or more days of the month.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>priority</td>
<td>The priority of the job.</td>
</tr>
<tr>
<td>procedureName</td>
<td>The name of the procedure to invoke.</td>
</tr>
<tr>
<td>processName</td>
<td>The name of the application process to invoke.</td>
</tr>
<tr>
<td>scheduleDisabled</td>
<td>If this box is &quot;checked!&quot;, the schedule will run. You can disable this schedule whenever necessary.</td>
</tr>
</tbody>
</table>
### Name | Description
---|---
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/workflow (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/workflow (don’t start a new job after this time); time values use a 24-hour clock, for example, 17:00.
tierMapName | The name of the tier map used to determine where to run the process.
tierResourceCounts | Resource count per resource template tier. (The alternate argument name ‘tierResourceCount’.)
timeZone | The time zone to use when interpreting times.
weekDays | Days of the week: 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

A version of an application with specific artifact versions and the state of the application at any point in time.

#### Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td>snapshotName</td>
<td>Name of the snapshot that must be unique within the application.</td>
</tr>
</tbody>
</table>

#### Optional Arguments
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentVersions</td>
<td>Component names and version used for snapshot. Use keyword 'LATEST' to indicate latest version. (The alternate argument name is 'componentVersion'.)</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of environment from which snapshot be created.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an object that is being renamed.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- **property**

**stage**

A logical grouping of pipeline tasks

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td>stageName</td>
<td>The name of the stage.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>afterStage</td>
<td>If specified, the stage will be placed after the named stage.</td>
</tr>
<tr>
<td>beforeStage</td>
<td>If specified, the stage will be placed before the named stage.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an object that is being renamed.</td>
</tr>
<tr>
<td>pipelineName</td>
<td>The name of the pipeline.</td>
</tr>
</tbody>
</table>
DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- task
- property

stateDefinition

A state definition in a workflow definition. Each workflow can contain one or more states.

Required Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name used for the state definition</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameters</td>
<td>The actual parameters to the state definition's process. (Alternate argument name 'actualParameter')</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>True if the state definition should remove all actual parameters to the process.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>startable</td>
<td>True if the workflow can begin in this state.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>The name of the sub procedure.</td>
</tr>
<tr>
<td>subproject</td>
<td>The name of the project that contains the sub procedure.</td>
</tr>
<tr>
<td>substartingState</td>
<td>The name of the starting state to use in the subworkflowDefinition.</td>
</tr>
<tr>
<td>subworkflowDefinition</td>
<td>The name of the subworkflowDefinition.</td>
</tr>
</tbody>
</table>
**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- `formalParameter`
- `emailNotifier`
- `property`
- `transitionDefinition`

**step**

A unit of logic that will execute on an agent.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>proceduresName</code></td>
<td>Name for the procedure that must be unique within the project.</td>
</tr>
<tr>
<td><code>projectName</code></td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td><code>stepName</code></td>
<td>Name of the step that must be unique within the procedure.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>actualParameters</code></td>
<td>Actual parameters passed to an invoked subprocedure (Alternate argument name 'actualParameter')</td>
</tr>
<tr>
<td><code>alwaysRun</code></td>
<td>True means this step will run even if preceding steps fail in a way that aborts the job</td>
</tr>
<tr>
<td><code>broadcast</code></td>
<td>True means replicate this step 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).</td>
</tr>
<tr>
<td><code>clearActualParameters</code></td>
<td>True if the step should remove all actual parameters.</td>
</tr>
<tr>
<td><code>command</code></td>
<td>Script to execute the functions of this step; passed to the step's shell for execution.</td>
</tr>
<tr>
<td><code>condition</code></td>
<td>A fixed text or text embedding property references that is evaluated into a logical TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE.</td>
</tr>
<tr>
<td><code>credentialName</code></td>
<td>The name of the credential object.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>errorHandling</td>
<td>Specifies error handling for this step.</td>
</tr>
<tr>
<td>exclusive</td>
<td>True means the resource acquired for this step will be retained for the exclusive use of this job. This means 2 things: first, no other job will be able to use that resource, regardless of its step limit, until this job completes; second, 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.</td>
</tr>
<tr>
<td>exclusiveMode</td>
<td>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.</td>
</tr>
<tr>
<td>logFileName</td>
<td>Name of the log file for a step; specified relative to the root directory in the job's workspace.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>parallel</td>
<td>True means this step and all adjacent steps with the flag set will run in parallel.</td>
</tr>
<tr>
<td>postProcessor</td>
<td>This command runs in parallel with the main command for the step; it analyzes the log for the step and collects diagnostic information.</td>
</tr>
<tr>
<td>precondition</td>
<td>A fixed text or text embedding property references that is evaluated into a logical TRUE or FALSE. An empty string, a &quot;0&quot; or &quot;false&quot; is interpreted as FALSE. Any other result string is interpreted as TRUE.</td>
</tr>
<tr>
<td>releaseExclusive</td>
<td>True means the resource acquired for this step will be no longer be retained for the exclusive use of this job when this step completes.</td>
</tr>
<tr>
<td>releaseMode</td>
<td>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.</td>
</tr>
<tr>
<td>resourceName</td>
<td>Name for the resource that must be unique among all resources.</td>
</tr>
</tbody>
</table>
Using the ElectricFlow DSL

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shell</td>
<td>Name of the shell program that will execute the command and postprocessor for the step.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>Name of a procedure to invoke during this step.</td>
</tr>
<tr>
<td>subproject</td>
<td>Name of the project containing the procedure to invoke during this step.</td>
</tr>
<tr>
<td>timeLimit</td>
<td>Maximum amount of time the step can execute; abort if it exceeds this time.</td>
</tr>
<tr>
<td>timeLimitUnits</td>
<td>Units for step time limit: seconds, minutes, or hours.</td>
</tr>
<tr>
<td>workingDirectory</td>
<td>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.</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- `emailNotifier`
- `property`
- `property`

**task**

A representation of task within a stage or gate.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td>taskName</td>
<td>The name of the task</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameters</td>
<td>Actual parameters passed to an invoked subprocedure (Alternate argument name 'actualParameter')</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>afterTask</td>
<td>If specified, the task will be placed after the named task.</td>
</tr>
<tr>
<td>approvers</td>
<td>A list of task approvers who receive the notification. (Alternate argument name 'approver')</td>
</tr>
<tr>
<td>beforeTask</td>
<td>If specified, the task will be placed before the named task.</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>True if the task should remove all actual parameters.</td>
</tr>
<tr>
<td>credentials</td>
<td>Credentials to be used in the task. (Alternate argument name 'credential')</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>enabled</td>
<td>True to enable the task.</td>
</tr>
<tr>
<td>environmentName</td>
<td>Environment name to create from template.</td>
</tr>
<tr>
<td>environmentTemplateName</td>
<td>Environment template name.</td>
</tr>
<tr>
<td>errorHandling</td>
<td>Specifies error handling for this task.</td>
</tr>
<tr>
<td>gateType</td>
<td>The type of the gate.</td>
</tr>
<tr>
<td>keepOnError</td>
<td>True to keep environment on error (default is false)</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>notificationTemplate</td>
<td>String containing email formatting instructions for generating notifications.</td>
</tr>
<tr>
<td>pipelineName</td>
<td>The name of the pipeline</td>
</tr>
<tr>
<td>skippable</td>
<td>True if a task is skippable.</td>
</tr>
<tr>
<td>snapshotName</td>
<td>Name of the snapshot associated with the application.</td>
</tr>
<tr>
<td>stageName</td>
<td>Name of the stage to which this task belongs to.</td>
</tr>
<tr>
<td>startTime</td>
<td>The time of day to begin invoking this task (24-hour clock, for example, 17:00).</td>
</tr>
<tr>
<td>subapplication</td>
<td>The name of the application that owns the subprocess.</td>
</tr>
<tr>
<td>subprocedure</td>
<td>If referencing a procedure, the name of the procedure.</td>
</tr>
<tr>
<td>subprocess</td>
<td>The name of the process.</td>
</tr>
</tbody>
</table>
Using the ElectricFlow DSL

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subproject</td>
<td>If referencing a procedure, the name of the procedure’s project.</td>
</tr>
<tr>
<td>taskProcessType</td>
<td>The type of the process a task can invoke.</td>
</tr>
<tr>
<td>taskType</td>
<td>The type of the task.</td>
</tr>
<tr>
<td>tierResourceCounts</td>
<td>Resource count per resource template tier (Alternate argument name ‘tierResourceCount’)</td>
</tr>
<tr>
<td>workspaceName</td>
<td>The name of the workspace.</td>
</tr>
</tbody>
</table>

**DSL Methods for ElectricFlow Objects That Can Be Nested Inside**

- `emailNotifier`
- `property`
- `property`

**tierMap**

A map to hold mappings between application and an environment tiers.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationName</td>
<td>The name of the application</td>
</tr>
<tr>
<td>environmentName</td>
<td>The name of the environment.</td>
</tr>
<tr>
<td>environmentProjectName</td>
<td>The name of the environment's project name.</td>
</tr>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>applicationEntityRevisionId</td>
<td>Revision ID of the versioned object</td>
</tr>
<tr>
<td>tierMapName</td>
<td>The name of the tier map. If not specified, ElectricFlow uses a hyphenated name consisting of the application and environment names.</td>
</tr>
</tbody>
</table>
**transitionDefinition**

Defines how a workflow must transition from one state to another.

**Required Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td>stateDefinitionName</td>
<td>The name used for the state definition.</td>
</tr>
<tr>
<td>transitionDefinitionName</td>
<td>The name used for the transition definition.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameters</td>
<td>The actual parameters to the transition’s target state.</td>
</tr>
<tr>
<td>clearActualParameters</td>
<td>If this is set to true or 1, the transition should remove all actual</td>
</tr>
<tr>
<td></td>
<td>parameters from the target state.</td>
</tr>
<tr>
<td>condition</td>
<td>A fixed text or text embedding property references that is evaluated</td>
</tr>
<tr>
<td></td>
<td>into a logical TRUE or FALSE.</td>
</tr>
<tr>
<td></td>
<td>If the string is empty or this is set to false or 0, the text is</td>
</tr>
<tr>
<td></td>
<td>interpreted as FALSE.</td>
</tr>
<tr>
<td></td>
<td>If the string is not empty or this is set to true or 1, the text is</td>
</tr>
<tr>
<td></td>
<td>interpreted as TRUE.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by</td>
</tr>
<tr>
<td></td>
<td>ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
</tbody>
</table>
Using the ElectricFlow DSL

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetState</td>
<td>Target state for the transition definition.</td>
</tr>
<tr>
<td>trigger</td>
<td>Specifies the type of trigger for this transaction.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>userName</td>
<td>The user name.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>Email address of the user.</td>
</tr>
<tr>
<td>fullUserName</td>
<td>Full name of the user.</td>
</tr>
<tr>
<td>groupNames</td>
<td>List of groups that this user is in.</td>
</tr>
<tr>
<td></td>
<td>An alternate name is <code>groupName</code>.</td>
</tr>
<tr>
<td>migrateSettings</td>
<td>Migrate the user or group settings to this user name.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an object that is being renamed.</td>
</tr>
<tr>
<td>password</td>
<td>The user's password.</td>
</tr>
<tr>
<td>removeFromAllGroups</td>
<td>If this is set to <code>true</code> or <code>1</code>, ElectricFlow removes this user from all groups.</td>
</tr>
<tr>
<td>sessionPassword</td>
<td>Session user's password.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectName</td>
<td>Name for the project that must be unique among all projects.</td>
</tr>
<tr>
<td>workflowDefinitionName</td>
<td>The name of the workflow definition.</td>
</tr>
</tbody>
</table>

Optional Arguments

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an object that is being renamed.</td>
</tr>
<tr>
<td>workflowNameTemplate</td>
<td>The template used to name instances of this workflow definition.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>workspaceName</td>
<td>The workspace name.</td>
</tr>
</tbody>
</table>

Optional Arguments
Using the ElectricFlow DSL

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agentDrivePath</td>
<td>On Windows, the path name to the root directory of a workspace, specified with a drive letter.</td>
</tr>
<tr>
<td>agentUncPath</td>
<td>On Windows, the path name to the root directory of a workspace, specified with a UNC path.</td>
</tr>
<tr>
<td>agentUnixPath</td>
<td>On UNIX, the path name to the root directory of a workspace.</td>
</tr>
<tr>
<td>credentialName</td>
<td>The name of the impersonation credential to attach to this workspace.</td>
</tr>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>local</td>
<td>True if the workspace is 'local'.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an existing object that is being renamed.</td>
</tr>
<tr>
<td>workspaceDisabled</td>
<td>True means this workspace is disabled.</td>
</tr>
<tr>
<td>zoneName</td>
<td>The zone name.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zoneName</td>
<td>The zone name.</td>
</tr>
</tbody>
</table>

**Optional Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Comment text describing this object that is not interpreted at all by ElectricFlow.</td>
</tr>
<tr>
<td>newName</td>
<td>New name for an object that is being renamed.</td>
</tr>
</tbody>
</table>
DSL Methods for ElectricFlow Objects That Can Be Nested Inside

- property

Troubleshooting and FAQs

Troubleshooting

<table>
<thead>
<tr>
<th>Issue</th>
<th>What to Do</th>
<th>Links to Related Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Get help on the DSL methods</td>
<td>To get the complete list of supported DSL methods, enter the following command: ectool evalDsl --describe 1</td>
<td>Getting Started with DSL on page 760</td>
</tr>
<tr>
<td></td>
<td>To get help on a particular DSL method, enter the following command: ectool evalDsl &lt;dsl_method_name&gt; --describe 1</td>
<td>Using the ElectricFlow Perl API on page 40.</td>
</tr>
<tr>
<td></td>
<td>Example: ectool evalDsl procedure -- describe 1</td>
<td></td>
</tr>
<tr>
<td>The DSL script completes successfully using the evalDsl command; however, the ElectricFlow objects are not created or updated as expected.</td>
<td>Use the debug option to trace and debug the script processing using the following command: ectool evalDsl &lt;dsl&gt; --debug 1</td>
<td>evalDsl on page 670</td>
</tr>
<tr>
<td></td>
<td>The debug option will allow evalDsl to generate debug output that you can use to follow the DSL script processing by evalDsl and identify any possible issue with the DSL script.</td>
<td>Creating and Running DSL Scripts on page 760</td>
</tr>
</tbody>
</table>

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`
ElectricFlow

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:

```groovy
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('COMMANDERHttps_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:

```xml
<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]/[types]/[artifact]-[revision](-[classifier]).[ext]"/>

</filesystem>

<ibiblio name="localm2" root="file:${user.home}/.m2/repository/"
checkmodified="true" changingPattern=".*" changingMatcher="regexp" m2compatible="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:

```ruby
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
```